2016

A Novel Magnetics and IMU Based Design for Untethered and Hands-Free Human Computer Interaction

Stephen J. Norris
University of Rhode Island, stephen_norris@my.uri.edu

A NOVEL MAGNETICS AND IMU BASED DESIGN FOR UNTETHERED AND HANDS-FREE HUMAN COMPUTER INTERACTION

BY

STEPHEN J. NORRIS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2016

MASTER OF SCIENCE THESIS

OF

STEPHEN J. NORRIS

APPROVED:

Thesis Committee:

Major Professor Qing Yang

Steven Kay

Bongsup Cho

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2016

ABSTRACT

The current status of Human Computer Interaction (HCI) stands to develop greatly, with many new technologies offering promising new methods interacting with a computer. The goal of this project is to develop a novel device that is compatible with desktop style point-and-click interfaces designed for mice but has the unique advantages of being 'Untethered' and 'Hands-free'. The use of Electromyography (EMG) and an Inertial Measurement Unit (IMU) was initially investigated, leading to an alternative design utilizing a series of small Hall Effect sensors and corresponding magnet(s). The cursor response was identified as the most important factor to ease of use, with physiological tremor being the main source of disturbance. Without friction to cancel physiological tremor, countering algorithms were developed to improve ease of use. A final prototype was developed as part of testing and for proof of concept.

ACKNOWLEDGMENTS

The group of professors I chose to be part of my committee should come as no surprise. They are all premier professors, and I am fortunate for the relationship I have had with them. First I would like to acknowledge Dr. Qing Yang for providing the inspiration for this project as well as the support and guidance throughout the process to complete it. It was also through his research grant that I was provided funding to support attending graduate school. Secondly I would acknowledge Dr. Steven Kay for both being an excellent teacher, bringing the difficult and often misunderstood topics of statistical signal processing to bear in a way that was clear and concise, as well as providing sound advice when I was unsure how to approach a problem. Thirdly I would like to acknowledge Dr. Bongsup Cho, who has been as excellent mentor and advisor throughout my academic career.

I would also be remiss if I did not acknowledge the other students who assisted in my endeavors. Primarily I would like to thank Timothy Forbes for his work in providing the inspiration for the project and for taking time to provide me with his materials and personal support with the EMG designs. I would echo the same thanks to our mutual friend Tom DeRensis, another graduate student, for together with Tim inspiring me to pursue my master's degree. Additionally I would like to thank Robert Hernandez for his assistance with developing the SVM algorithm and in troubleshooting the EMG designs. I would also like to thank Tanya Wang for her assistance in the physical manufacture of the prototype. Last, but not least, I would especially like to thank my family for their continued support and encouragement.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure **Page**

CHAPTER 1

Introduction

1.1 Overview and Motivation

The current status of Human Computer Interaction (HCI) stands to develop greatly, with many new technologies such as computer vision promising new interaction with a computer. However, the most recent major breakthrough in HCI can be seen in the widespread application of touchpads and touchscreens for mobile computing such as in smartphones and tablets. However the standard computer mouse is still in widespread use for desktop computers. A possible explanation is that for a new device to enter mainstream use it must both be compatible with existing user interfaces, and provide a unique advantage to its use in those existing interfaces. A new device could prove its value using existing interfaces even though it could support more advanced methods of user input. Later, if interest is garnered, more advanced interfaces could be developed to take advantage of the additional ability. The goal of this project is to develop a novel device that is compatible with desktop style point-and-click interfaces designed for mice, but has the unique advantages of being 'Untethered' and 'Hands-free'. A prototype device was created and tested which utilizes a combination of two types of sensors, an Inertial Measurement Unit (IMU) and a series of small Hall Effect sensors and corresponding magnet(s).

To further understand why new technologies may not be accepted, one can imagine a sensor that can track the user's hand and individual finger movements. While such a device may get some attention from electronics media for being novel, there is no readily available interface in current use that could make effective use of such data. One could create a simplified design where a single finger is tracked as the mouse cursor, with finger flexing indicating clicking actions. However doing

so would not take advantage of all the extra data that the sensor provides beyond what a normal mouse is capable of. Without the extra data being used the device does not provide any other strong merit to entice purchase.

On the other hand, one can examine how the touchscreen has entered widespread use. Especially for mobile computing, the touchscreen boasts a great advantage from being built into the screen and not being an external device that must be carried around in addition to the main device. Before modern smartphones reached markets touchscreens could be found in occasional use as effectively a multi-layer interactive button panel. However, because touchscreens had been widespread, but not common use, when modern smartphones were developed the use of a touchscreen became obvious. In addition, advancements in touchscreen technology making them easier to use ensured their adoption. However, as noted by Bogdanoff [1] in an article about the rise of touchscreens, "It is not enough to simply add a touch screen to an existing UI. Trying to navigate awkward menus and select small icons will quickly cause you to go back to the mouse and keyboard." Smartphone operating systems and applications have been designed from the ground up to support the touchscreen interface. Only now with their success are touch interfaces entering the realm of personal computing devices.

1.2 Objective and Scope

Therefore with these factors in mind the objective of this project is to create a new design for HCI that is compatible with the standard mouse interface, but provides advantages over the standard mouse. The device would also provide the possibility of more advanced input that could be exploited should interfaces to harness the additional functionality be developed. The main advantages of the device will be referred to as 'untethered' and 'hands-free'. The term untethered refers to the property of the device not restricting where the user can use the

device, due to a fixed sensor, optical sensor, or in the case of the standard desk, the need for a flat surface. The device achieves this by simply being worn by the user, and not requiring any other sensors that might tether the user. Hands free refers to the fact that even though the device is worn there are no buttons, switches, or other objects that must be placed in the user's hand to use the device's main functionality. Though the prototype device created requires the user to wear a glove, the use of magnets and Hall Effect magnetic sensors means that there is no physical touching required. The Hall Effect sensors are placed on the back of the palm, and the corresponding magnets are simply secured to the glove where they need to be. The advantage this provides is that to stop using the device and use the driving hand the device simply just needs to ignore what the user is doing with that hand. This can be extremely advantageous when rapidly switching between pointing and typing, as the device proved to be able to automatically detect when the user switches to typing and temporarily shut off until the user is finished. In addition, this advantage works with the untethered property as the user does not need to keep track of a device that is held, since the device can be worn at all times without restricting hand use. There exist "Air Mice" which are untethered devices but have the downside that the user must hold the device, and place it in their pocket or such when not in use and moving around. It has the same disadvantage of the TV remote that is just never in reach because the user left it somewhere away from the couch.

In terms of testing the performance of the device, it is difficult to provide a simple metric to judge performance that can be compared to existing devices. The more important factor in evaluating the device's success is ease of use. One of the biggest challenges in developing the device proved to be disturbance in the user's pointing response resulting likely from Physiological tremor. Physiological

tremor is a much lesser issue for a standard mouse because friction between the mouse and the surface it is placed on prevents weak disturbances from affecting the cursor position. This is important for most point-and-click interfaces as movement between the triggering of a mouse button press and release correspond to dragging. Without friction to filter out small disturbances it is nearly impossible to trigger a simple click instead of a dragging input. Therefore the biggest factor is not the true accuracy of the device and performance which could be easily measured, but whether it provides a similar filtering effect while also feeling properly responsive.

List of References

[1] D. Bogdanoff. "The touch screen revolution." Feb. 2015. [Online]. Available: http://www.techbriefs.com/component/content/article/ntb/features/feature-articles/21545

CHAPTER 2

Literature Review and Analysis

2.1 Literature Review

The idea of a glove mouse is not a new idea as there exist designs of this type already. A prime example is "Computer mouse on a glove" [1] which details a simple glove design that uses pressure plates and external sensor for tracking. This design is a tethered design due to the use of an external sensor for tracking the glove. Another example is the complex "Magic Mouse"[2] which details a double-handed multi-component system including two different sets of inertial sensors, and a number of contact sensors. While this device is an untethered design as it uses sensors placed on the user, it is not hands-free because of the contact sensors which are bulky and cover the palm of the user's hand. The authors do mention the untethered nature of the device, but the device seems to lend itself to full-time use due to the complexity and fullness of the design. Being hands free was not part of the design goal of this device.

The current state of the art for technology with these possible design goals involves Electromyography (EMG). There are two papers detailing similar devices that employ Electromyography (EMG) to perform pattern recognition of gestures in combination with an inertial measurement unit.[3][4] Often abbreviated IMU, an inertial measurement unit is a sensor which typically combines various other sensors such as accelerometers, gyroscopes, and magnetometers to provide measurements related to the inertial frame of the sensors involved. The use of EMG means the sensors are placed on the forearm, leaving the user's hand completely unobstructed. In fact, it may be possible to be used by hand amputees. The first paper, "A Novel HCI based on EMG and IMU"[3] is the earliest example showcasing such a device. The paper is unclear on many aspects of the design but essentially an

accelerometer is used to move the cursor while the EMG system is used as a gesture system to trigger events such as clicking. The second paper, "Mouse HCI through combined EMG and IMU" by Forbes[4], which showcases a similar device served as a starting point for this project. This design aimed to combine the inputs of the EMG and IMU in a way that would improve the performance of the device. To illustrate, movement in the upward direction would be ignored unless the EMG pattern recognition system indicated the user was performing a gesture of the wrist rotating upwards.

In addition to these devices, a recent commercial device called Myo[5] has become available that utilizes the same kind of EMG and IMU design. However the device is not being marketed and sold as a mouse replacement, but as a gesture input platform with a programming interface available for developers to add support for the device. As such the device is only as useful if there is an available application for what the user wants. It appears that there is no standard pointing application as a separate group has published a paper detailing their design for a pointing interface, called Myopoint[6]. A key point is that in comparison to Forbes they use a separate click and pointing interface for the most part. However they do detail three filtering or correcting techniques they used to deal with false positives from the EMG system and IMU system that take advantage of inter system design, but not to the extent of Forbes.

In addition to previous devices it became apparent that information regarding muscle physiology might be useful, specifically regarding Physiological Tremor. As the device is attached to the user's arm which will move independent of friction, the effect of tremors become much more important. While there are specific kinds of tremors associated with disorders such as Parkinson's, Physiological Tremor is a naturally occurring tremor that everyone exhibits. Its effect can be magnified

and easily observed by placing a sheet of paper on one's hand. Fortunately there is much available literature on the subject, varying from studies attempting to design tremor-canceling devices for delicate surgery[7] to various studies investigating its causes[8].

2.2 Analysis of Literature

As previously mentioned the Forbes paper[4] served as the starting point for the project. A different IMU and the same but new Trigno wireless EMG system[9] were obtained in order to try to replicate the same results. Difficulties with the EMG pattern recognition system became obvious very quickly. Caution should be taken when reviewing pattern recognition papers as many will report a high accuracy, which may seem to be excellent. However, in practice a high accuracy such as 99%, combined with an update rate of 100Hz will mean an average of one false classification per second. In addition, these accuracies are often the cross-validated training accuracy obtained from a set of data recorded in a short period of time. If the underlying random processes, perhaps such as muscle fatigue, are not stationary over the long term, a model obtained with that short-time segment of data could start to fail spectacularly later. In addition, the training data itself is not a valid representation of the full feature space that would be observed during actual use. Even with the Myo system[5], which is a proprietary system promising high accuracy, false positives can be present. Specifically, the Myopoint paper details: "Moving the arm quickly or extending it very far can activate forearm muscles leading to false-positive hand postures recognized by the EMG system"[6]. Personal experimentation with the EMG system would often obtain a fairly high training accuracy only for the system start to fail later. Majority vote and other such averaging systems were used but the reality is that they do nothing to prevent errors if the random process changes over a long period of time, or when the

features are observed outside the range of training. Furthermore, the EMG system required each wireless electrode to be pasted individually to the correct position on the skin to obtain the signal from a specific muscle. Variations in the exact placement of the electrode, adhesion, and other factors related the EMG signals mean that each time the electrodes are put on the system needed to be retrained. For pattern recognition techniques, Linear Discriminant Analysis (LDA), which was used in the aforementioned paper, and Support Vector Machines (SVM) were implemented. Both algorithms resulted in the same difficulties, but SVM did provide better accuracies. Further investigation also revealed that the individual electrodes of the Trigno wireless system were leaking electrical interference into the signals as shown in Figure 1. The presence of these signals could be the cause of difficulties. Because the electrodes are extremely sensitive it is possible that slight differences in sensor orientations or positions when recording the various classes of the training data could change the signals in a way that the pattern recognition techniques would incorrectly use those differences to differentiate the classes. Eventually the idea to use small inexpensive hall effect sensors to replace the EMG system arose and focus shifted away from using this technology.

Investigating Physiological tremor led to some varying data but most studies pointed to a peak oscillation around 8-12Hz[10]. To compare to the literature spectra of data recorded from various combinations of IMU position and support using the IMU chosen for the design. The obtained data showed a clear peak at about 9Hz with the IMU strapped around the palm and with the forearm supported (Figure 2). While this is not useful for the following design this is similar to what is reported in the literature. However, other combinations did not all show such a clear result. This is shown later in the overall plot detailed in Figure 6 as part Chapter Three. Lakie et al. suggests that one of the main

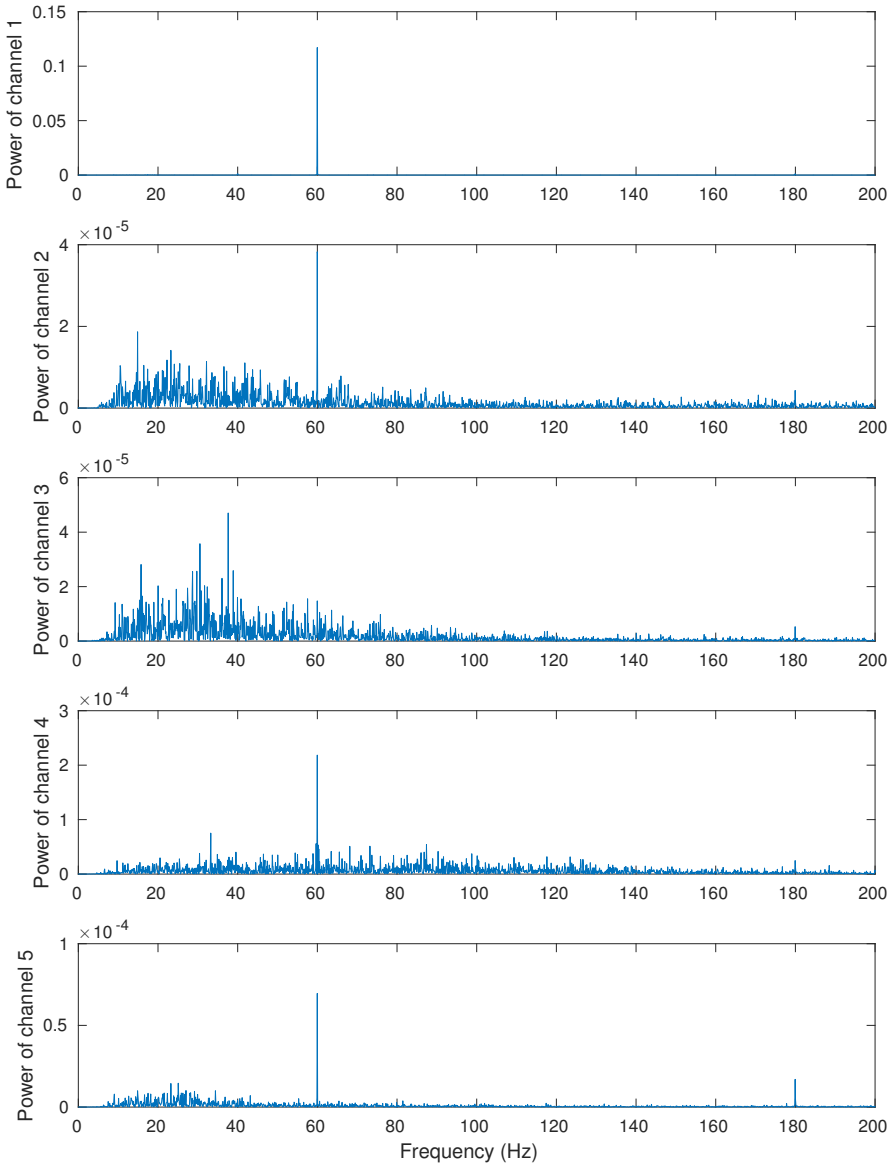

Figure 1: Power Spectral Density plots for each of the five EMG channels from a set of training data in initial experiments. Each channel is on of five Trigno wireless sensors recorded through the digital interface.

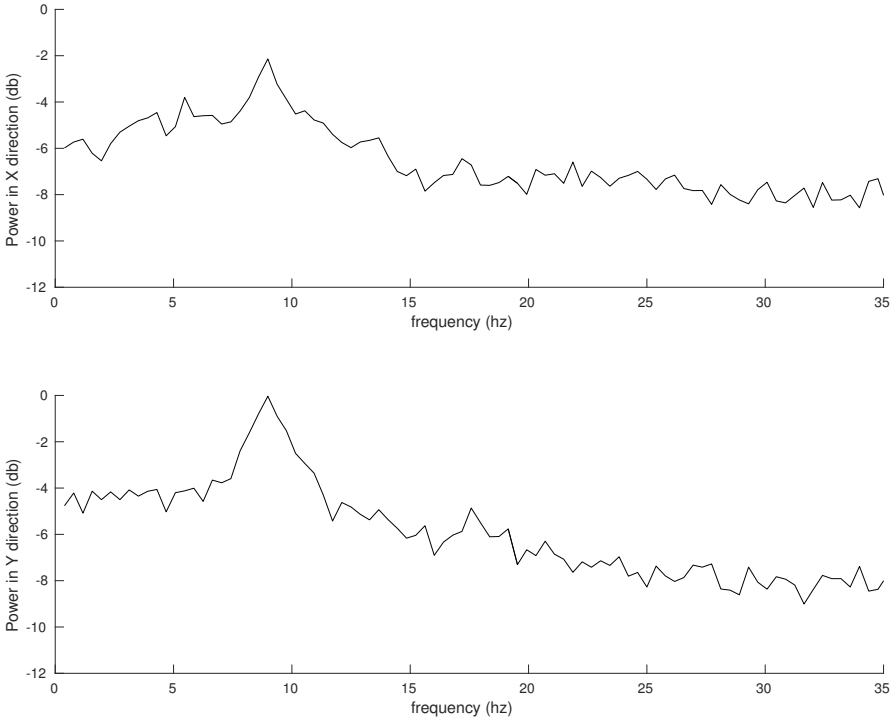

Figure 2: Power Spectral Density of data recorded using the IMU mounted on the back of the palm with forearm support

factors of Physiological tremor is resonance of the bone and muscle structure[8]. Looking at the the spectra one might hypothesize that the combinations involving better support display clearer peaks by isolating specific joint's resonances. The other effect is that combinations without support are more chaotic in the same way that a multi-segment pendulum can swing wildly. Further study towards this hypothesis is far outside the scope of this project, but it is clear that the main source of noise and disturbance to the cursor position is an inherent phenomenon of the human body and not the IMU itself.

List of References

[1] M. Bajramovic, "Computer mouse on a glove," July 2003, uS Patent App. 10/382,849. [Online]. Available: http://www.google.com/patents/US20030137489

[2] E. L. Sean Chen. Cornell University. "Mister gloves - a wireless usb gesture input system." 2010. [Online]. Available: https://courses.cit.cornell.edu/

ee476/FinalProjects/s2010/ssc88_egl27/

[3] A. Xiong, Y. Chen, X. Zhao, J. Han, and G. Liu, "A novel hci based on emg and imu," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, Dec 2011, pp. 2653–2657.

[4] T. Forbes, "Mouse hci through combined emg and imu," Master's thesis, University of Rhode Island: Open Access Master's Theses Paper 43, 2013. [Online]. Available: http://digitalcommons.uri.edu/theses/43

[5] Thalmic Labs. "Myo gesture control armband - wearable technology by thalmic labs." 2015. [Online]. Available: https://www.myo.com/

[6] F. Haque, M. Nancel, and D. Vogel, "Myopoint: Pointing and clicking using forearm mounted electromyography and inertial motion sensors," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 3653–3656. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702133

[7] C. Riviere, R. S. Rader, and N. V. Thakor, "Adaptive canceling of physiological tremor for improved precision in microsurgery," *IEEE Transactions on Biomedical Engineering*, vol. 45, no. 7, pp. 839 – 846, July 1998.

[8] M. Lakie, C. A. Vernooij, T. M. Osborne, and R. F. Reynolds, "The resonant component of human physiological hand tremor is altered by slow voluntary movements," *The Journal of Physiology*, vol. 590, no. 10, pp. 2471–2483, 2012. [Online]. Available: http://dx.doi.org/10.1113/jphysiol.2011.226449

[9] Delsys Incorporated. "Trigno lab — wireless emg system." 2015. [Online]. Available: http://www.delsys.com/products/wireless-emg/trigno-lab/

[10] L. L. Rubchinsky, A. S. Kuznetsov, V. L. W. MD, and K. A. Sigvardt, "Tremor," *Scholarpedia*, vol. 2, no. 10, p. 1379, 2007, revision 135551.

CHAPTER 3

Materials and Methods

3.1 EMG and IMU precursor

The design process began by first attempting to develop a device similar to the one from "Mouse HCI Through Combined EMG and IMU"[1] to be used as a baseline. The EMG system[2] used was the same, but the system was interfaced differently. In the previous design the analog output mode of the EMG system was used and fed to an analog to digital converter system connected to the computer running the main system. Instead, the digital output mode was used, which transfers directly to a computer digitally over USB. The Vectornav rugged Vn100 IMU[3] was chosen instead of the Xsens wireless IMU. It was decided to use a wired IMU due to the cost difference between wired and wireless IMUs. Additionally the Vn100 was chosen as it can negate drift with its onboard sensor fusion algorithms, and boasts a complete selection of output modes. Physically, the rugged model of the Vn100 comes in a very small metal encasing, only approximately two centimeters on each side and less than a centimeter wide, making it the perfect size for mounting on the arm. In addition, a separate header was available that would match the output plug for making a custom connector when needed. Otherwise, the IMU could be used via a serial USB interface. To physically interface with the hall effect sensors and the IMU an mbed NXP LPC1768 microcontroller was used.[4]

3.2 EMG processing

Pattern recognition techniques were used to process the EMG data. Two techniques were investigated, Linear Discriminate Analysis (LDA) and Support Vector Machines (SVM). The muscles targeted were the Extensor Carpi Radialis,

Flexor Carpi Ulnaris, Flexor Carpi Radialis, Extensor Carpi Ulnaris, and Flexor Digitorum Superficialis. As in Forbes[1], the classes enumerated are rest, up, down, left, right, left click, and right click. The features used for dimensional reduction are Mean Absolute Value (MAV), Waveform length, Turns, and Zero Crossings. Due to the difference of recording process, the EMG signals do not have the same scale as in the Forbes paper. Because the Turns and Zero Crossings had specific thresholds which relied on a specific scale, new thresholds had to be set and the output characteristics could be different. In addition, the EMG signals over the digital interface are available at a rate of two kilohertz, as opposed to one kilohertz. This resulted in increasing the window increment and length. The window increment went from 20 samples (20 milliseconds at 1kHz) to 40 samples, and window length went from 60 samples to 120 samples.

The overall process is as following: a recording program was used to save a portion of EMG data to a series of files corresponding to each class. These files were then given to a training program which would output the final model file for the live program to use for classification. Cross validation was used for training both LDA and SVM. For SVM the radial basis function kernel was chosen and it was implemented using a third-party library. For both, majority vote was used to filter the resulting classifications.

3.3 IMU processing

The IMU was interfaced with by using the USB serial communications library written in C++ provided by Vectornav[3]. The Quaternion output was chosen for obtaining the orientation of the IMU as it was the simplest output type which would allow custom taring to be done in software. Each incoming quaternion sample is converted to a rotation matrix which represents the rotation as a linear transformation and used for further processing. The columns of the matrix

correspond to the forward direction, horizontal sideways towards the user's right, and downwards. Each sample matrix is first adjusted by the taring system detailed in the next section.

3.3.1 Taring

The taring system is used to allow setting a point of reference that further measurements are made from. This is similar to zeroing a scale before taking a measurement. To create a taring system a zeroing rotation matrix is calculated. This matrix is the inverse of the yaw rotation when the system is tared. Multiplying this matrix against each incoming matrix makes the output relative to the original matrix in terms of the yaw orientation. The pitch and roll orientations are unaffected as their zero points do not change. The algorithm to obtain this matrix is to essentially flatten first column of the tare matrix, and to set the third column to straight up. To illustrate this, consider the following example rotation matrix which represents an arbitrary rotation around the third axis:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In this example, the third coefficient of the first column is zero, but this is not the case with full rotational freedom. However, one can simply zero out that value to obtain a vector in the correct orientation. This has the same effect of simply projecting the forwards direction of the IMU into the horizontal plane. Since rotation matrices should be orthonormal, the resulting vector will need to be normalized to ensure this. Filling out the rest of the new rotation matrix is easy as the second column is simply a rearrangement and negation of the first column. Finally, the whole matrix needs to be inverted, which is easily done with a simple negation of coefficients corresponding to the odd sine function. The resulting

matrix, given the normalized vector $\begin{bmatrix} x_1 & x_2 & 0 \end{bmatrix}^T$, the final matrix is as follows:

$$\begin{bmatrix} x_1 & x_2 & 0 \\ -x_2 & x_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

3.3.2 Translating 3-Dimensional data to Screen Coordinates

For the IMU rotation to be used to drive cursor movement a scheme for translating rotation to (**X**,**Y**) screen coordinates is needed. The simplest scheme is that the yaw angle corresponds to the x direction in screen space and the pitch angle corresponds to the y direction in screen space. The yaw direction works fine in practice but the pitch direction ends up feeling awkward at high angles, especially with elbow flexion. An alternative scheme was devised by imagining how a laser pointer moves when pointed at a flat surface like a wall. When the laser's ray is perpendicular to the wall, a change in angle corresponds roughly linearly to a change in position. However, as the angle approaches parallel, the same change in angle corresponds to a greater and greater change in position, approaching infinity.

To actually realize this idea where the user actually uses a surface is not possible as all the position related measurements from the IMU are relative. Nor is this even desired due to the untethered design goal. Instead the design should emulate there being a virtual wall of a certain distance away. It turns out what is needed is to simply get the tangent of the pitch. This is just the ratio of up to horizontal of the first column of the rotation matrix. If we represent that column as $\begin{bmatrix} \overline{x} & \overline{y} & \overline{z} \end{bmatrix}^T$ then the mapping ends up as follows:

$$\mathbf{Y} = d\frac{z}{\sqrt{x^2 + y^2}} \tag{2}$$

The constant d would be the distance to the wall, but really just serves as a scaling coefficient. However, previously the scheme was simply $\mathbf{Y} = pitch$, with

pitch in degrees. It would be useful for the two schemes to match in scaling, at least when close to zero pitch. This led to the following equation:

$$\mathbf{Y} = pitch = d\tan(\frac{pitch\pi}{180}) \tag{3}$$

Setting pitch to zero doesn't work quite as well as it simply it is the algebraic equivalent of throwing the equation into a black hole. However setting pitch to one, which is quite close since pitch is in degrees, works quite easily:

$$1 = d\tan(\frac{\pi}{180}) \tag{4}$$

$$d = \frac{1}{\tan(\frac{\pi}{180})} \approx 57.3 \tag{5}$$

As a quick observation, one can note that at the value of d is equal to $\mathbf{Y}$ when the pitch is 45 degrees.

Finally, one minor nuance to dealing with the $\mathbf{X}$ value is needed. Specifically it is calculated nominally as following:

$$\mathbf{X} = \frac{180 * \arctan 2(y, x)}{\pi} \tag{6}$$

Ultimately the problem with this is that the arctan function is periodic when it should not be. Instead of this, the previous yaw value is stored and the difference to the new yaw value is added using the following equation:

$$\mathbf{X}[n] = \mathbf{X}[n-1] + \mathrm{mod}(yaw[n] - yaw[n-1] + 540, 360) - 180 \tag{7}$$

3.3.3 Using Roll

While yaw and pitch rotations primarily drive cursor movement, the sensor also can provide measurements of roll rotations. This was found to be highly useful due to a clear difference between the roll orientation from a natural pointing pose to the typing pose. This meant that input could be temporarily disabled automatically when the user goes to type, and reactivated as soon as the user

returns to pointing. This feature is quite similar in use to palm-recognizing features designed to disable the trackpad on a laptop when the user is typing, but is much more reliable. In addition, rolling the forearm over, as with wrist supination, is unnatural while pointing, and can be used to quickly and reliably indicate something to the system. It was decided to use this action to switch the device into sleep mode. This was useful as it also performed well as an emergency-stop in case the device started to produce undesired input. It was also decided that the condition of the pitch angle being extremely high or low, beyond the range of comfort, would activate the same effect.

Calculating the current roll from the current rotation matrix is more complex than calculating yaw. Representing the rotation matrix as $\begin{bmatrix}\overline{x} & \overline{y} & \overline{z}\end{bmatrix}$, the important vectors are x and either y or z. As x represents the direction of pointing, the rotation of y and z around it corresponds to the roll angle. In this case the vector z is chosen for the calculation. First, the goal is to remove the pitch rotation from the vectors. We can zero out part of the x vector and renormalize to make the next step simpler.

$$\overline{p} = \frac{\begin{bmatrix}x[0] & x[1] & 0\end{bmatrix}^T}{\sqrt{x[0]^2 + x[1]^2}} \tag{8}$$

Next, use GramSchmidt orthogonalization to remove pitch from the z vector using the x vector. In this case normalization is not needed.

$$\overline{u} = \overline{z} - (\overline{p} \cdot \overline{z})\overline{p} \tag{9}$$

Now, set up two normalized two-element vectors to be used in the final step:

$$\overline{c_1} = \frac{\begin{bmatrix}p[0] & p[1]\end{bmatrix}^T}{\sqrt{p[0]^2 + p[1]^2}} \overline{c_2} = \frac{\begin{bmatrix}u[0] & u[1]\end{bmatrix}^T}{\sqrt{u[0]^2 + u[1]^2}}$$

Finally, the angle can be calculated via arctan using the adjusted z vector. The adjustments allow the x and y values needed for the arctan2 function to be obtained simply. The determinant calculation simply provides a value of -1 or 1, giving the

needed sign missing due to the always positive square root calculation.

$$roll = \frac{180 * \arctan 2(\left|\overline{c_1} \quad \overline{c_2}\right| \sqrt{u[0]^2 + u[1]^2}, u[2])}{\pi} \tag{10}$$

3.4 Hall Effect Sensor Processing

Developing the Hall Effect Sensor portion of the device turned out to be straightforwards. The main issue was that there is some sort of periodic interference somewhere in the ADC pathway. The design plan was to create a simple detector to detect whether the magnet was present near any of the hall effect sensors or not. The final design ended up being essentially that but with a different threshold for activating and deactivating, as other than the periodic interference there were no other complications.

3.4.1 Interference

The interference observed is a periodic signal with a long component and a short component, with an overall period of about 20 seconds. Each channel of the ADC board was tested and all channels exhibited this, but with each channel having a very small, but noticeable time offset. This interference spanned a larger range than the actual apparent sensor noise so eliminating or reducing it would provide much better sensitivity. Filtering capacitors were added to try to reduce noise but due to the extremely slow period of the main interference it did not have much effect. It is likely the cause is within the ADC board itself. On the signal processing end one could imagine creating an estimator to estimate the time offset and amplitude of the interference and cancel it out, but given the long period, this is unwieldy. Ultimately the solution came in the form of simply placing a dummy sensor on the breadboard, not to sense any magnets, but to just record the interference. Each channel has almost the same signal but with a very small time offset so it worked near perfectly in the longer component, showing the most

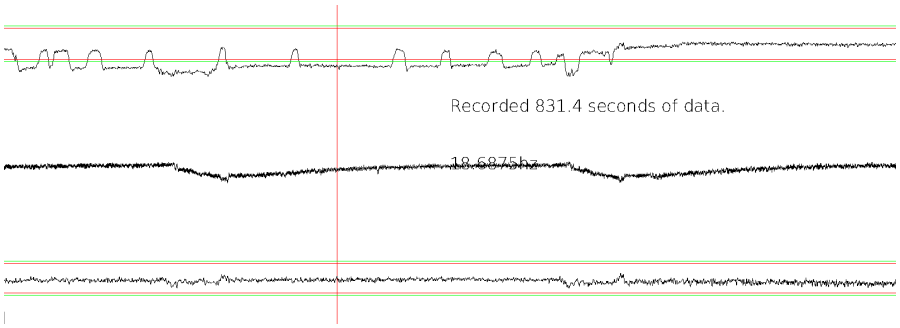

Figure 3: Screen capture from a live display utility showing the Hall Effect signals as reported in real-time with a working prototype version. The center signal is the reference signal that is to cancel the interference. The top signal exhibits some clicking activity. In addition, the green horizontal lines show the activation thresholds, with the red horizontal lines showing the deactivation thresholds.

discrepancy during the short components. This can be seen in figure 3. With this modification the effect is reduced, and afforded a much tighter set of thresholds and higher sensitivity was achieved.

3.4.2 Detector design

As the interference signal was much larger than any apparent sensor noise, building a noise model for a detector simply involved recording data from the sensors. Thresholds were then set using the maximum or minimum values recorded, depending on which was relevant. Doing so is essentially setting the desired probability of false alarm to zero, which would normally adversely affect detection performance. However, since the noise signal is pretty well behaved, the thresholds stay tight enough so that this works absolutely fine. In addition it was decided to use a different threshold depending on the previous state of the detector. That is, when there is no magnet present, the threshold used for the next sample is a little higher, requiring a stronger signal to be a detection. This is primarily in case the signal drifts near the normal threshold, where oscillations due to noise could cause the signal to go above and below repeatedly. For this kind of system a detector that changes state rapidly is undesirable as it will trigger unintended mouse clicks.

3.5 Managing Physiological Tremor

Ultimately the performance of the pointing system comes down to two things: being able to move the cursor finely while maintaining responsiveness, and the ability to hold the cursor still. The main culprit affecting performance in both these cases turns out to be a phenomenon known as Physiological Tremor. It is typically characterized by peak oscillation around a specific frequency, typically around 7 to 11Hz. While there are many types of tremors with various clinical significance such as Parkinsonian tremor, Physiological tremors are present in all humans. Furthermore, there is evidence to support the theory that one of the main mechanisms behind it is physical resonances of the muscular and bone structures of the human body. In this way, the apparent oscillation is not a result of muscle firing at a specific frequency but rather the muscular and bone structure resonating at that frequency. If it was the case that muscle firing contributed to causing these oscillations one might expect to be able to observe the muscle firing in order to cancel out the disturbances. Instead, the muscular and bone system under question probably acts more along the lines of a car's radio antenna vibrating wildly when driving down the highway. Additionally there is evidence that muscle contraction, as during dynamic movement, changes the frequency of oscillation[5]. Altogether this is a daunting source of noise and disturbance to the pointing system.

In the case of allowing fine control while maintaining responsiveness, there is a trade off in terms of an arbitrary scaling factor between the actual user's movement and the corresponding number of pixels that the cursor moves. Being able to adjust this scaling factor is a standard feature for any computer and more or less exists due to differences in screen resolutions and the user's preference. For example, a way of setting this scaling value would be to determine how far the mouse must move to traverse the entire screen. The distance needed should fit somewhat comfortably

within the space the user has to move in. On the other hand, one could set the scaling around being able to do fine pointing accurately by reducing the sensitivity. This reduces the effect of disturbances and allows the user to make much larger movements which can be done more accurately. However, doing so can reduce the responsiveness of the cursor, meaning the user might have to pick up the mouse in order to cross the entire screen. One feature that attempts to strike a balance is mouse acceleration, where stronger motion is amplified, allowing the user to shoot the mouse across the screen but still maintaining fine control for weaker motion. However, the nature of the acceleration can be somewhat arbitrary to the user and ultimately is up to the user's preference. Applying such a response curve to the motion of the cursor for this design as a default should not be done without due process and reason.

The second case, being able to hold the cursor still, is easily achieved by normal mice due to friction. However, for this design the reality is that the user cannot hold the cursor perfectly still on their own. The lack of friction means even the slightest disturbance or twitch changes IMU's velocity instead of being canceled out. The ideal design for this would be to emulate the effect of friction using the input from the frictionless system. This is challenging, because while one could simulate friction acting on the raw cursor movements, that would cause the cursor to stop while the user is still moving the IMU. This disconnect is jarring to the user and in practice just following the raw cursor movement during large IMU movement works fine. The trick is to only affect the cursor movement under weaker movements, where friction might stop the cursor from movement.

3.5.1 Precursor Design

The first attempt at a solution to this challenge was essentially a sensitivity curve. The curve mapped the raw delta in cursor position in the x and y directions

independently to a new value of reduced magnitude. This is essentially taking the idea of mouse acceleration but going the other direction and dampening smaller movements. The curve can be described by the following function:

$$\mathbf{X}'[n] = \mathbf{X}[n](1 - e^{2b(1-|\frac{\mathbf{X}[n]}{a}|)})u(|\mathbf{X}[n]| - a) \tag{11}$$

The parameter a is half the width of the 'deadzone' of this function, where the function equals zero due to the unit step function on the right side of the equation. This is an important feature of the curve as its intent is to allow the cursor to be held still if the user can keep their movements within the deadzone. The parameter b controls the drop off speed and shape of the function. It was set to one, because at that value the second derivative is zero at the edge of the deadzone. A example of this plot can be found later in figure 5 in the first column. Since the parameters are not defined, suitable values would have to be determined in testing. However a problem was quickly apparent. While it was possible to set the deadzone to be wide enough that the cursor could be held still, fine cursor control became impossible. If the user wishes to make a small adjustment, they must move quick enough to surpass the deadzone, but only just over. This region of the curve has a high slope, which means that the user has to be much more accurate than normal. Errors in movement are amplified by a value equal to the slope. Making small adjustments accurately is challenging enough without this added difficulty.

3.5.2 Main Design Part One: Gaussian Based Sensitivity Curve

The next attempt at a solution involved more foresight. First, a data record was taken of the cursor's movement while the user attempted to simply hold the IMU still. The result is shown in figure 4. The distribution of the cursor's velocity calculated by frame by frame difference follows closely to a bivariate Gaussian distribution. Of course, this ignores the reality that each velocity sample is correlated, as the velocity does not change quickly. However recording the velocity

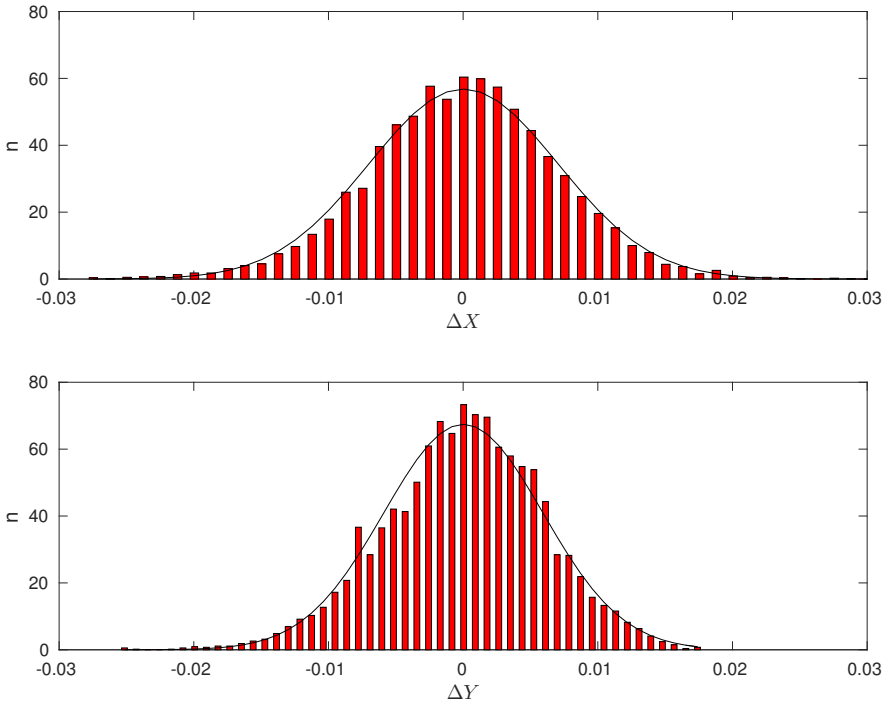

Figure 4: Histogram showing distributions of measured difference in raw X and Y cursor locations

over a long period of time provides a distribution of the velocities that can be expected at any single time. That the distribution turned out to be bivariate Gaussian with zero mean and covariance is also fortunate as the probability distribution function (given $\mathbf{x} = \begin{bmatrix} X[n] & Y[n] \end{bmatrix}^T$) can be rewritten as:

$$p(\mathbf{x}) = \frac{1}{2\pi\sqrt{|\Sigma|}} e^{-\frac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x}} = \frac{1}{2\pi\sqrt{|\Sigma|}} e^{-\frac{1}{2}r^2} \tag{12}$$

This causes the probability density function to essentially be an exponential probability density. Instead of the independent X and Y directions, the distribution uses the Mahalanobis distance, the value of r^2. This can be imagined as the radial distance (squared) from the center of the distribution if the variance in the X and Y directions are equal and are uncorrelated. More specifically, when the covariance matrix is determined via recording, this value corresponds to how powerful a movement is compared to the normal strength of the user's recorded tremors. This gives a great statistic for doing any sort of design. Furthermore, the

cumulative density function in terms of the Mahalanobis distance is as follows:

$$F(r^2) = 1 - e^{-\frac{r^2}{2}} \tag{13}$$

Now if we model the user's movement as either choosing to move an unknown amount, or choosing to not move, we can derive a Bayesian minimum mean square error estimator for the user's intended movement. Given the following posterior PDF:

$$p(\mathbf{m}|\mathbf{x}) = P\{no\ movement\}\delta(\mathbf{m}) + P\{movement\}\delta(\mathbf{m} - \mathbf{x}) \tag{14}$$

The value of the two probabilities can be restated with the CDF using the Mahalanobis distance ($r^2 = \mathbf{x}^T\Sigma^{-1}\mathbf{x}$):

$$p(\mathbf{m}|\mathbf{x}) = (1 - F(r^2))\delta(\mathbf{m}) + F(r^2)\delta(\mathbf{m} - \mathbf{x}) \tag{15}$$

$$p(\mathbf{m}|\mathbf{x}) = (e^{-\frac{r^2}{2}})\delta(\mathbf{m}) + (1 - e^{-\frac{r^2}{2}})\delta(\mathbf{m} - \mathbf{x}) \tag{16}$$

Which using the techniques for a Bayesian Means Squared Error estimator detailed in Kay[6] finally leads to an equation somewhat similar to the precursor design:

$$\hat{\mathbf{m}} = \mathbf{E}[\mathbf{m}|\mathbf{x}] = (1 - e^{-\frac{r^2}{2}})\mathbf{x} \tag{17}$$

A comparison of this sensitivity function, the precursor sensitivity function and the sensitivity function used by the Myopoint design [7] is provided in figure 5. The important thing to note is that this sensitivity curve stands out because it is based on an actual measured distribution. The sensitivity curve attempts to reduce the overall error by reducing the movement based on how likely the observed movements was from intentional or unintentional movement by the user. This estimator provides an excellent solution to the first problem of obtaining finer pointer control while maintaining responsiveness. However, this estimator does not solve the second problem as there is no deadzone or other effect that would allow for the cursor to be held still.

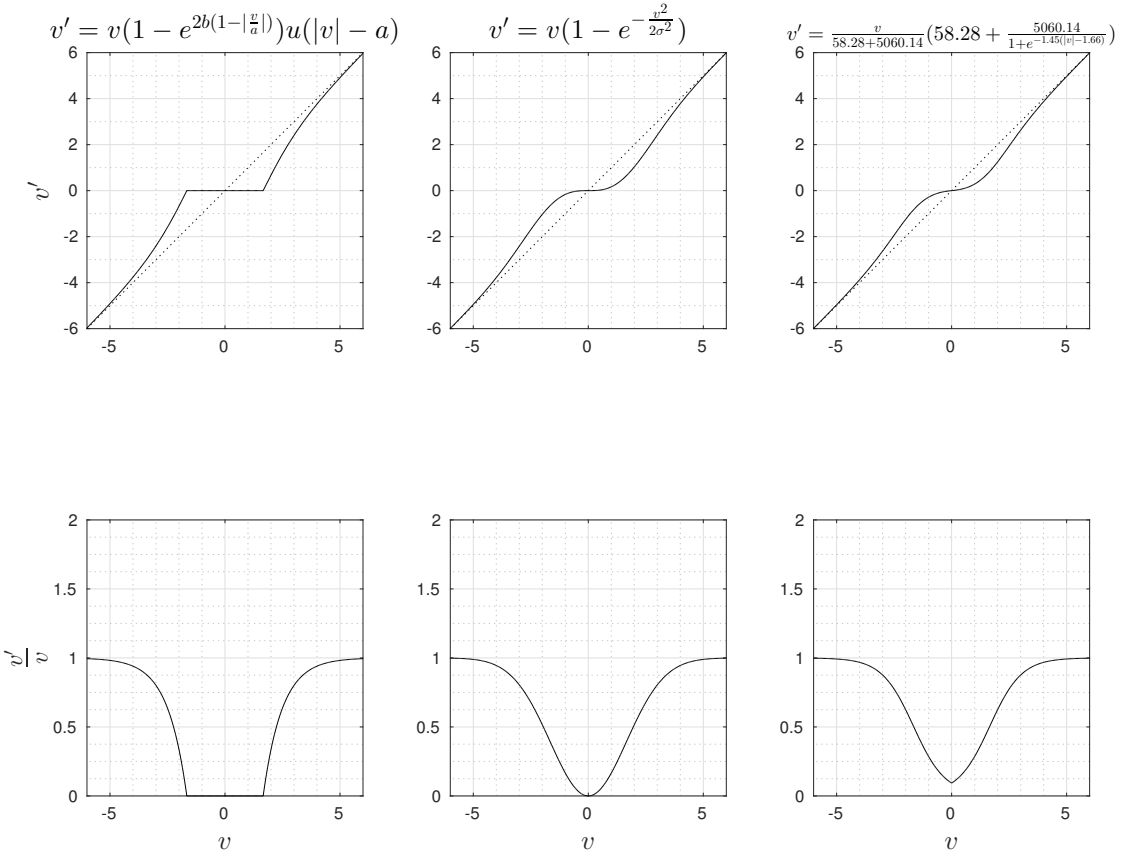

Figure 5: Comparison between different sensitivity curve functions. Shown in columns are from left to right: Precursor, Gaussian Based, Myopoint[7]. The Myopoint paper does not include absolute value of the velocity but it is assumed to be needed as otherwise the function is biased. The bottom row shows a normalized value, showing clearly the deadzone in the precursor design where the function zeroes out. The gaussian based design simply touches zero at a single point, but the curve from the Myopoint design does not actually reach zero. Functions have been parameterized to produce similarly scaled plots for comparions: $a = \sigma = 1.66$, $b = 1$.

3.5.3 Main Design Part Two: Simulated Friction

The second problem of being able to hold the cursor is still not solved and poses a more difficult challenge. Any reasonable solution is going to include some form of deadzone that causes the cursor to stop moving. The simple solution, just a flat deadzone, is not acceptable as it affects fine pointing control. Any design which applies a deadzone on all input is not a very good design as it makes it impossible to achieve small input. A modification would be to use a double threshold switching system with a high 'move' threshold and a low 'stick' threshold. Additionally while the sensitivity curve approach lends itself to processing on a sample-by-sample basis, a thresholding system could benefit from using more than one sample at a time. However, the analysis is more difficult as this introduces correlation between samples. Because the IMU is a physical object it must accelerate over time, resulting in each sample being fairly close to other recent samples. In addition there is the effect of physiological tremor which is typically characterized by oscillation.

In an attempt to understand the effects that this correlation might have, a set of data was recorded and analyzed to produce a series of power spectral densities (figure 6). Five combinations were recorded by simply recording the raw cursor signals using the working prototype system, with the IMU mounted at the back of the palm or on the wrist. For the case of the back of the palm, recordings were made with the entire arm unsupported, supported at the elbow, and supported midway on the forearm, isolating different joints from movement. For the wrist, recordings were made with and without the elbow support, leaving out forearm support. In addition a recording was made with the IMU simply sitting on the desk and not mounted on the arm.

As mentioned in chapter 2, the case of the palm mount with forearm supports shows the clearest peak. All of the mounted cases show significantly more

energy than the unmounted recording, which indicates that physiological tremor is the main source of disturbance and not sensor noise on the part of the IMU. Unfortunately the only case that shows a clear spectrum where a filter might be of use is the palm with forearm support, which was included to compare with other studies of physiological tremor, not because it would be a viable way to use this device. Mounting the IMU on the back of the palm and supporting the wrist would isolate the wrist as the joint to control the cursor. This would not be viable as the range of motion of the wrist is limited, compared to the elbow. A useful comparison however, is to compare the unsupported cases, to determine if mounting the IMU on the wrist or elbow affects the noise levels. In this comparison the wrist mount overall has slightly less noise. The wrist mount had been chosen originally on the intuition that it would be more stable, so no changes were made. The spectrum for this combination is shown in figure 7.

Unlike the setup of the IMU mounted on the palm with forearm support, the chosen setup has a smooth spectrum in comparison. It was decided to assume white Gaussian noise as it would simplify the problem and allow for some sort of solution to be devised. A possible design is to create a detector using a linear model of the cursor following a constant velocity. The model would be specified

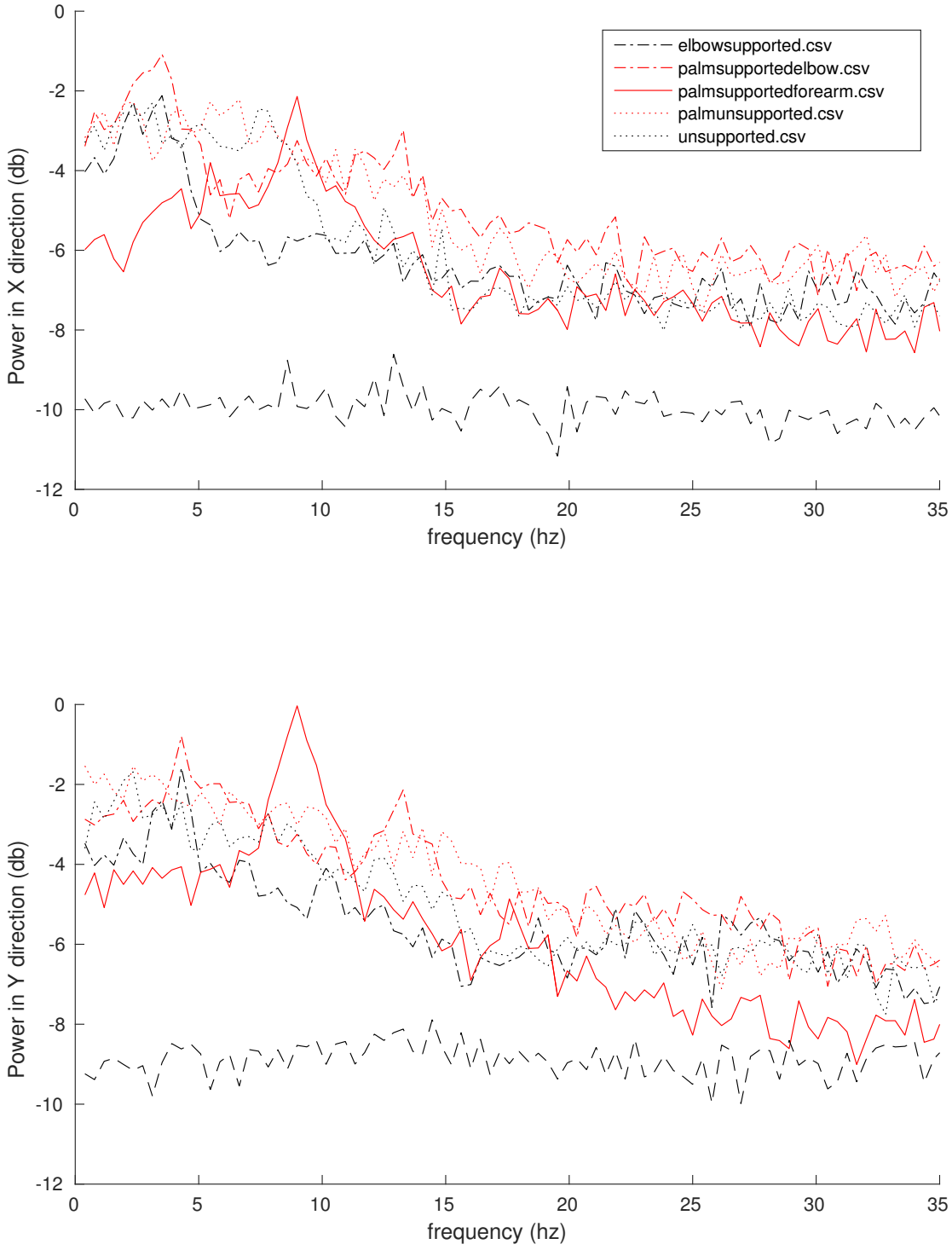

Figure 6: Power Spectrum overlay of various combinations of IMU position and support positions.

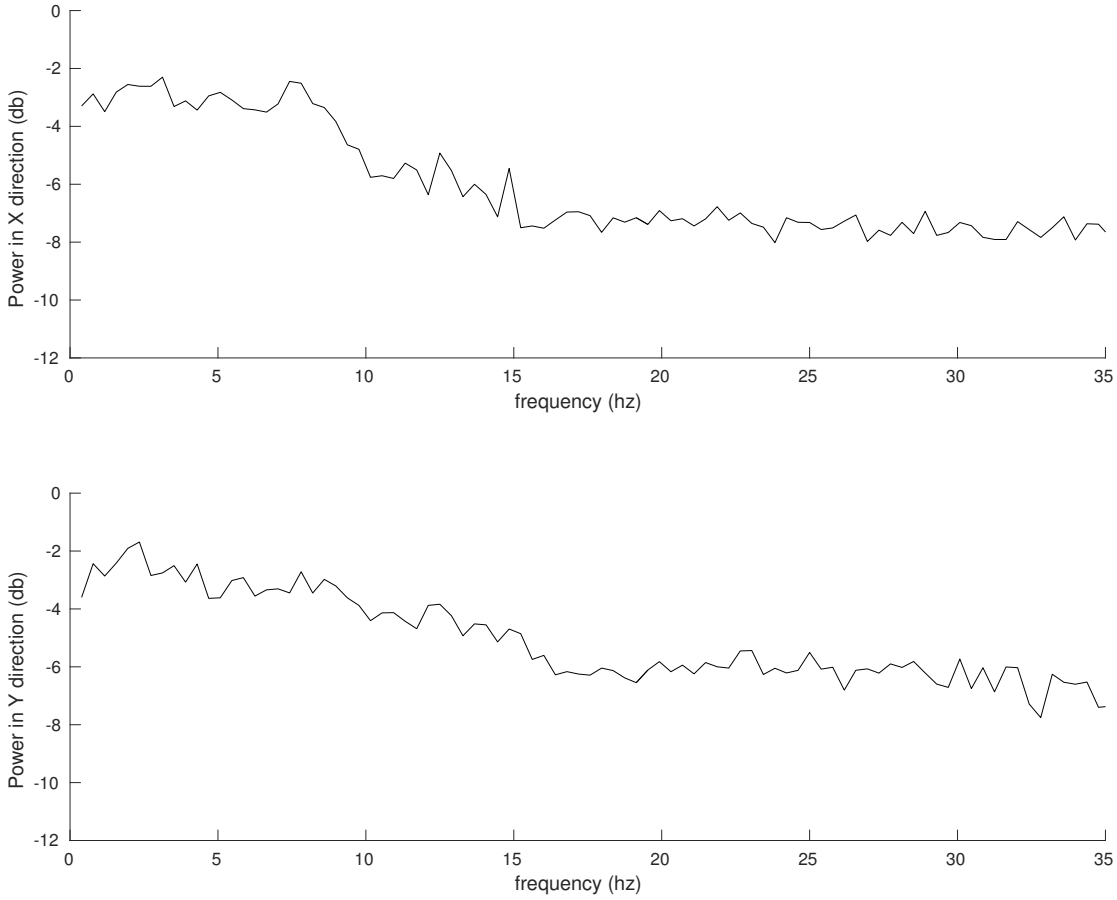

Figure 7: Power Spectrum of the IMU mounted on the wrist with no support, as used in the design.

as follows:

$$\mathbf{h} = \begin{bmatrix} -M \\ -M+1 \\ -M+2 \\ \vdots \\ M-2 \\ M-1 \\ M \end{bmatrix} \tag{18}$$

$$\mathbf{H} = \begin{bmatrix} \mathbf{h} & \mathbf{0} \\ \mathbf{0} & \mathbf{h} \end{bmatrix} \tag{19}$$

$$M = (N-1)/2 \tag{20}$$

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{21}$$

$$\mathbf{x} = \mathbf{H}\mathbf{v} + \mathbf{w} \tag{22}$$

With this linear model a generalized likelihood ratio test (GLRT)[6] can be easily derived:

$$\hat{\mathbf{v}} = \mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{x}$$

$$T(\mathbf{x}) = \sqrt{\hat{\mathbf{v}}^T\hat{\mathbf{v}}} > \gamma$$

Note that while one would normally include position parameters somewhere in a model of linear movement, $\mathbf{h}$ is set up to have an average of zero. This means that any constant vector will be orthogonal to $\mathbf{h}$, rendering any positional offset irrelevant to the calculation. There are two parameters still left unassigned however; the window length N, and the relevant thresholds to be used. In addition, the square root of the final test statistic is not necessary for the threshold test but is included so that the test statistic is the actual speed estimate. To this end, data segments were obtained featuring a strong, moderate, and weak single movement, to complement the usual 'noise' recordings. Using this data different values of N could be compared, and thresholds determined. The results of three window lengths are featured in Figures 8, 9, 10 and 11. For the sake of plotting, the strong

movement signal has been left out as they were off scale of the charts even when plotted at log scale.

A good window length to choose is one which strikes a good balance between smoothing the signals and being large enough to cause noticeable response delay. Of course, if the onset of a movement is strong enough the delay is shortened because the detector could be triggered before the new movement fills the window. Therefore the delay should cause the most impact for weak movements. With a 100Hz sample rate, a window length of 25 corresponds to a quarter second delay in the worst case. This value is no accident as it is roughly the normal human reaction time to visual stimulus[8].

Before deciding on the thresholds to use, there is an interesting observation to make: there is a characteristic difference between the noise signal and weak movement signal. While the weak movement signal remains flat, the noise signal rapidly dips sharply. This is likely due to how the test statistic turns the two dimensional velocity estimate into a single positive value. As physiological tremor is apparently due to a resonant effect, one might imagine the IMU as being on an flexible rod. If the rod oscillates back and forth, one would expect the velocity to be close to zero as the rod reaches a point of high stress and bounces back. These points where there is a sharp acceleration could easily cause such dips as the velocity vector quickly tracks near zero. Regardless of what is causing it, this characteristic provides an interesting choice when assigning thresholds. A double threshold switching system was chosen as with the Hall effect processing design. If a two threshold switching system is used, the stick threshold, such as noted by the red horizontal line in figure 10, the stick threshold can be set quite low. Although the signal is not constantly below the threshold, the rapid dipping means that the signal will pass under the threshold very often. Since this is a switching

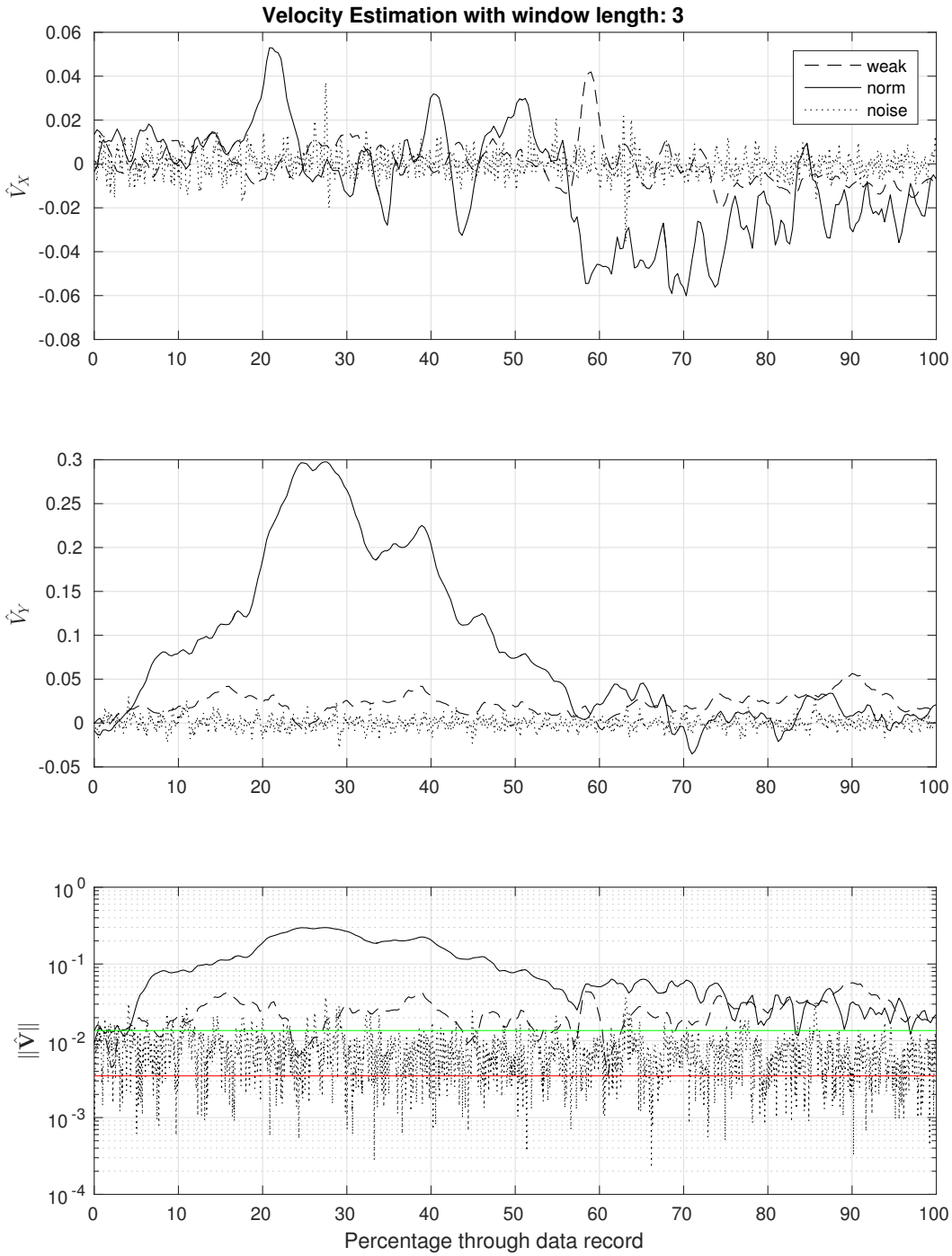

Figure 8: Velocity estimate signals of movement in the positive Y direction calculated offline on experimental data with $N_{win} = 3$

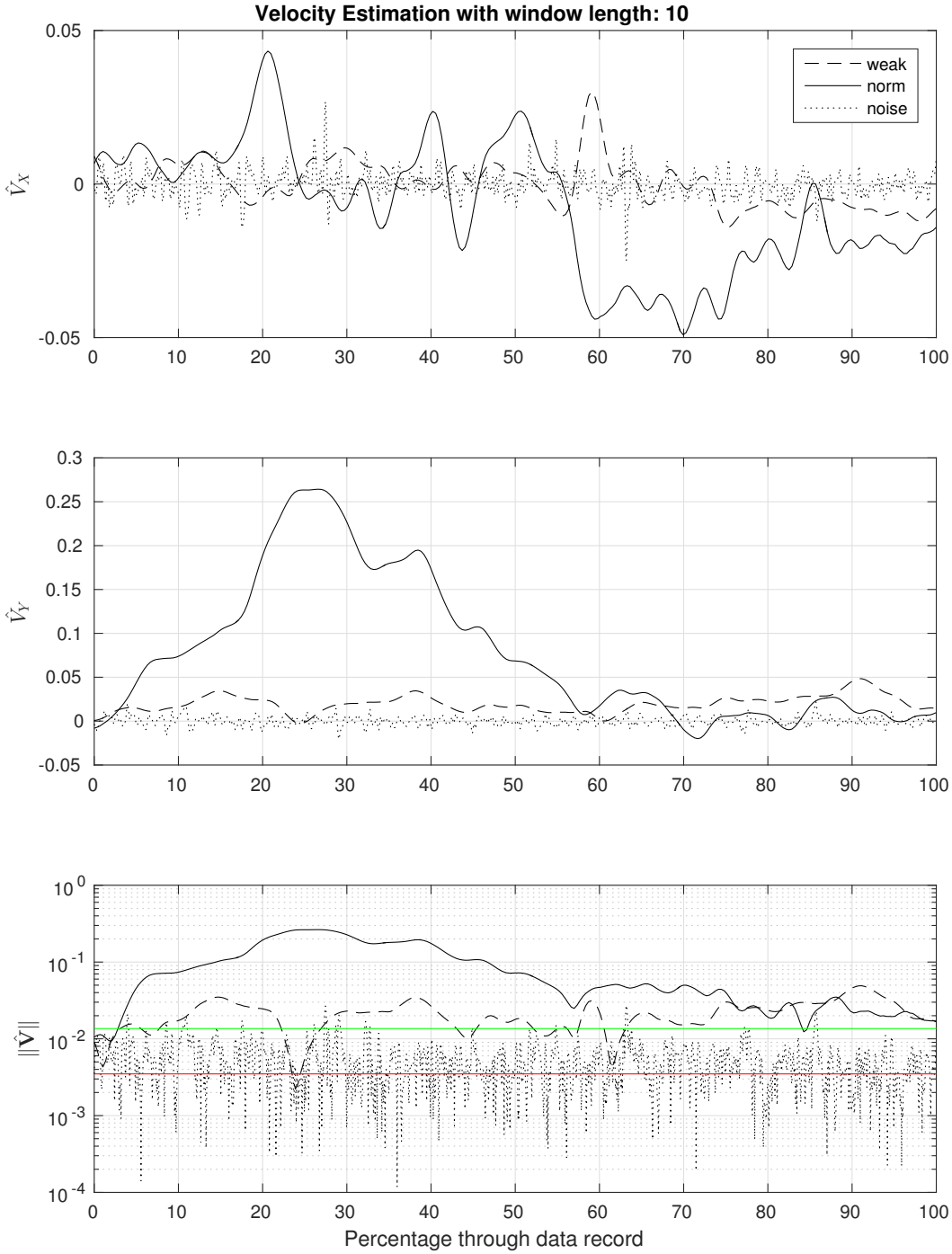

Figure 9: Velocity estimate signals of movement in the positive Y direction calculated offline on experimental data with $N_{win} = 10$

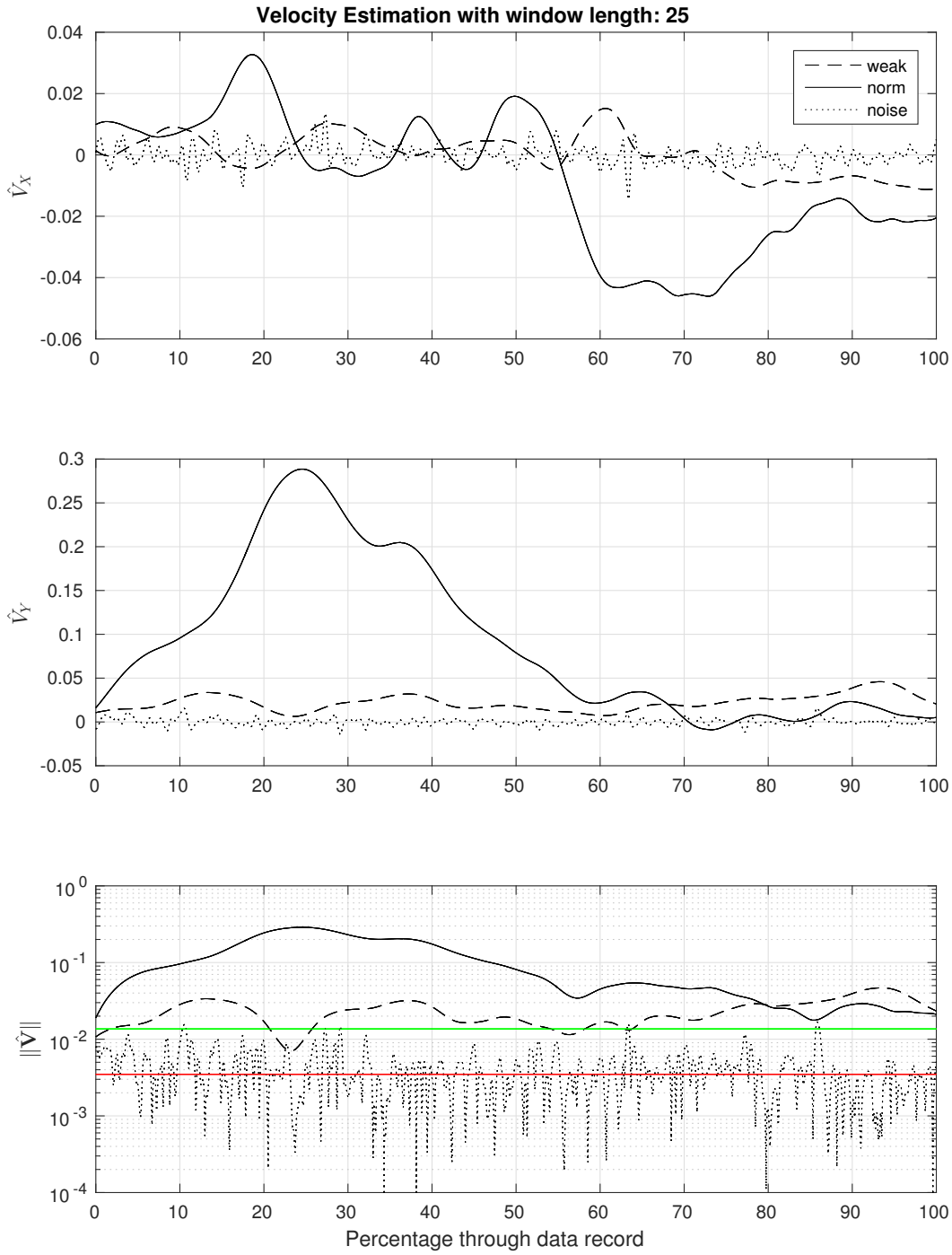

Figure 10: Velocity estimate signals of movement in the positive Y direction calculated offline on experimental data with $N_{win} = 25$

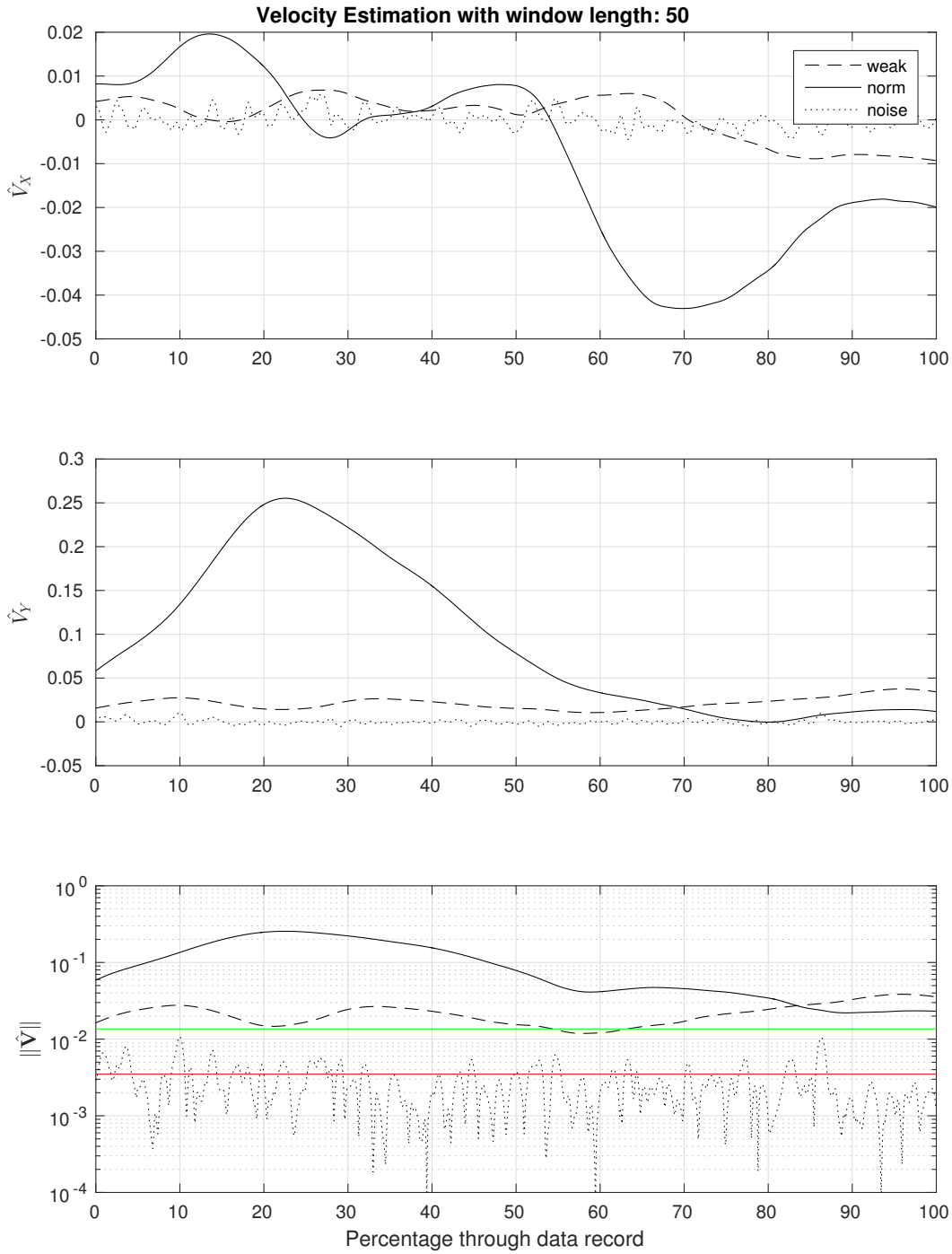

Figure 11: Velocity estimate signals of movement in the positive Y direction calculated offline on experimental data with $N_{win} = 50$

system, each of these dips will switch the system's state to stop the cursor from moving. As long as the threshold is not set too low, any potential delay may not be noticeable. Another consideration is that the user may cause a dip that will stick the cursor immediately at the end of their movement by actively arresting their own movement. The threshold start allowing movement again was simply set to the upper limits of the noise signal. This threshold should be set low to maintain responsiveness, but could be left to the user to adjust if they prefer. However, if set low enough, the user can start the cursor low enough in the sensitivity curve to still be able to make fine adjustments.

List of References

[1] T. Forbes, "Mouse hci through combined emg and imu," Master's thesis, University of Rhode Island: Open Access Master's Theses Paper 43, 2013. [Online]. Available: http://digitalcommons.uri.edu/theses/43

[2] Delsys Incorporated. "Trigno lab — wireless emg system." 2015. [Online]. Available: http://www.delsys.com/products/wireless-emg/trigno-lab/

[3] VectorNav. "Vn-100 rugged." 2015. [Online]. Available: http://www.vectornav.com/products/vn100-rugged

[4] NXP. "Arm mbed lpc1768 board." 2015. [Online]. Available: http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/lpc-cortex-m-mcus/lpc-cortex-m3/lpc1700-series/arm-mbed-lpc1768-board:OM11043

[5] M. Lakie, C. A. Vernooij, T. M. Osborne, and R. F. Reynolds, "The resonant component of human physiological hand tremor is altered by slow voluntary movements," *The Journal of Physiology*, vol. 590, no. 10, pp. 2471–2483, 2012. [Online]. Available: http://dx.doi.org/10.1113/jphysiol.2011.226449

[6] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[7] F. Haque, M. Nancel, and D. Vogel, "Myopoint: Pointing and clicking using forearm mounted electromyography and inertial motion sensors," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 3653–3656. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702133

[8] "Human benchmark." Mar. 2016. [Online]. Available: http://www.humanbenchmark.com/

CHAPTER 4

Conclusion

4.1 Results and Discussion

A prototype of the detailed design was created for the purpose of testing and proof of concept demonstration. The sensitivity curve and friction simulation designs were used in tandem. The Hall effect processing system was used with Hall effect sensors on the index and middle fingers, corresponding to left and right clicking as on a normal mouse. For the sake of prototyping, a modular system that

(a) Overall glove component with paperclip to show magnetism.

(b) Closeup of one of the Hall effect sensors

Figure 12: Photographs showing the glove components of the prototype

could be easily modified was used. Velcro strips were used to attach the IMU and a prototyping breadboard holding the microprocessor that used to record the Hall Effect signals and interface with the IMU. Each loop was created by taking two complementary strips of equal length and attaching the ends to each other, forming a loop. Each of three loops are then strapped tautly to the user's forearm. One at the wrist, to hold the IMU, and two on the forearm to hold the breadboard. This enabled relative ease of use for the prototype but such a design would likely be

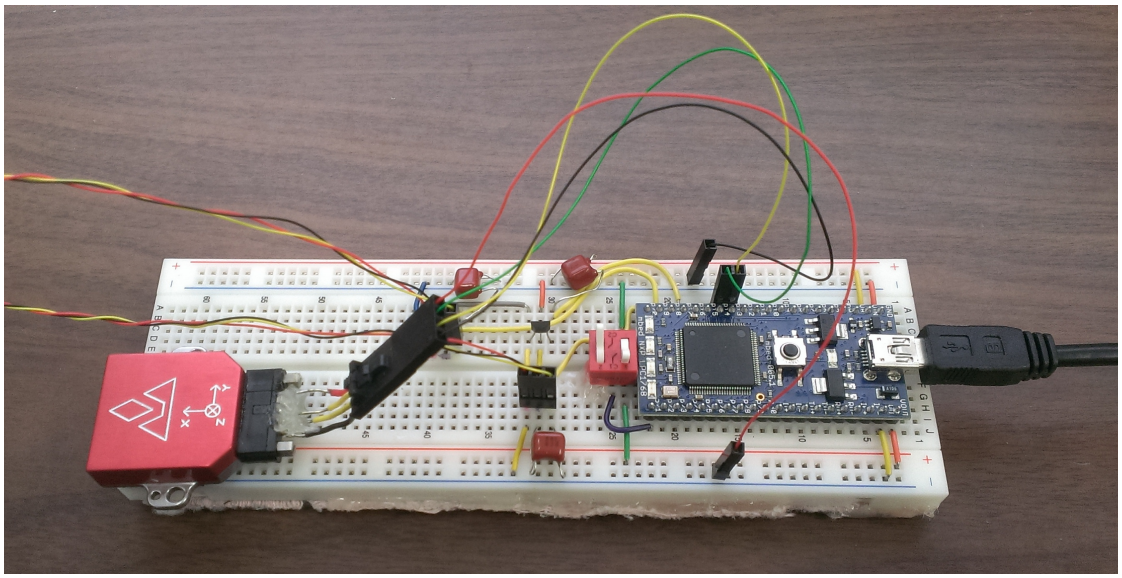

Figure 13: Prototyping breadboard showing various electronic components

irrelevant on a production design as most of the non-sensor electrical components could be miniaturized and placed on the back of the glove instead. The glove components were attached by use of a electronic glue gun. Low gauge flexible wires were used to allow flexibility between the Hall effect sensors, IMU, and breadboard.

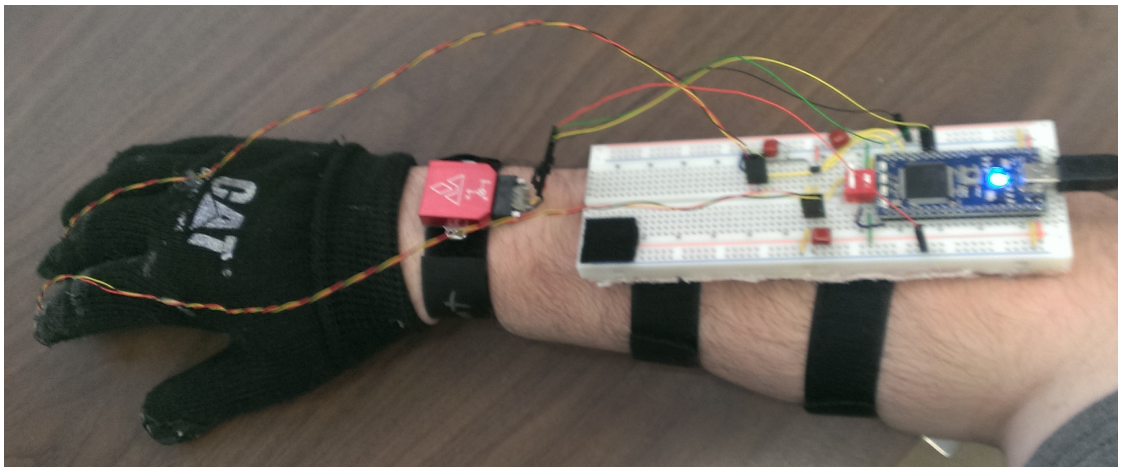

Figure 14: Overall view of prototype as when in use.

Testing the prototype was done by simply trying to use the device to perform the normal functions of a cursor. Early designs proved frustrating as the cursor

either remained unresponsive or could not be held still in order to successfully perform a click. However, the Hall effect design, once the interference was reduced proved to be reliable. Given the inexpensive nature of the sensors this was an effective alternative to utilizing EMG pattern recognition to achieve clicking control in terms of ease of use, reliability, and cost. The Hall effect design still allows the device to be hands-free by avoiding placing functionality such as buttons obtrusively inside the user's palm.

After incorporating the two main features of the IMU processing, the sensitivity curve and friction simulation, the device saw remarkable improvement. While the sensitivity curve was developed early, it alone did not solve the problem of the cursor constantly dancing around. Clicking properly was only possible by quickly 'tapping' and only in cases where dragging at the cursor's location was not possible. However, once the friction simulation was added, all these problems disappeared. The cursor sticking effect felt familiar and intuitive, given extensive experience with normal mice. Though the IMU may be moving, the actual muscle movement and cursor response feels natural.

4.2 Design Improvements

Though the prototype was successful as a proof of concept, it is not quite a complete device. There are some obvious improvements that would be included. In addition to the obvious wireless interface and battery power that would allow for a true untethered device, scrolling and additional button functionality was not developed for the prototype. However, extra button functionality could be added by including the ring finger, and by making use of the duality of magnetism. Each Hall effect sensor has the possibility of detecting a positive or negative magnetic field across its detector. The prototype simply detects any deviation from normal levels, not distinguishing between the different field directions. If the field direction

was considered the number of touch spots could be doubled, allowing for additional mouse buttons. A total of six should be theoretically possible, allowing for three additional mouse buttons corresponding to middle click, and backward and forward buttons. The sixth combination could be used to switch the device into 'scroll' mode, where the IMU's movement is used to scroll instead of moving the cursor, similar to the two-finger scroll common on touchpads.

Another possible design improvement would be to make use of Haptic technology. As such for the prototype the only indication that the user is clicking with the Hall effect sensors is that there is a corresponding response on the screen. This is not a substantial problem as the the system is very reliable and consistent, but it is possible that the use of some form of Haptic technique could improve the design. The touch response of pressing a button or clicking the mouse provides the user with feedback to confirm that their input is being accepted. However determining the most effective type device to use and exactly what kind of feedback to provide could be an entirely new study.

Besides these improvements, this device has additional potential that would require development of new interface designs. Because the IMU is not restricted to two-dimensional space, there is the potential to make use of a full six degrees of freedom movement between x, y, and z acceleration and yaw, pitch and roll rotations. For instance, to navigate a 3D virtual environment, one could 'grab' with the mouse and then move the IMU in the desired direction. Or when examining a 3D object, the full 3D rotation could be used to rotate the object on screen.

In addition to the obvious sensor advantages, the untethered and hands-free properties of the device provide entirely new possibilities of human computer interaction. The possibility of persistently carrying a wireless pointing device have not truly been explored. Wearable technology such as Google Glass or other mobile

technology such as smartphones could make use of this much more precise pointing technology. Smart televisions could make use of a desktop-style interface that could be used wirelessly. Remote control of devices such as quadrocopters would also be possible. A final possibility is that one could use two of the devices, one on each arm. Most people would not use two mice at once because it takes up space, but given the nature of this device, if one were operating in an environment where the hands-free nature is useful, it might also be useful operate with either hand. Additional applications may also be developed in which the ability to use two hands would be useful. In the end, this type of device stands to provide unique opportunities for modern human-computer interactions.

APPENDIX A

Source code for main application

In this Appendix source code for a C++ application is provided. This application is run on the user's computer. It provides a visual display of the hall effect sensor data, handles communicating with the microprocessor, and also handles actually sending input from the device to the computer system.

A.1 Hall effect processing object

This object abstracts the hall effect processing, automatically creating a data collection thread and handling functions such as checking clicking and drawing the signals.

```
/*
 * HallEffect.h
 *
 *  Created on: Jan 8, 2015
 *      Author: snorris
 */

#ifndef HALLEFFECT_H_
#define HALLEFFECT_H_

#include "IMU.h"
#include "math.h"
#define GLFW_INCLUDE_GLU
#include <GLFW/glfw3.h>
#include <GL/glc.h>
#include <eigen3/Eigen/Dense>
#include "kissfft/kiss_fftr.h"
#include <mutex>
#include <vector>
#include <thread>

class HallEffect {
public:
    const static int SAMPLERATE = 100, // rate of sensor update in Hz
        NWIN = 4, // window length of filter
        NFFT = 3*1600, // data length of fft window
        NFFTOUT = NFFT / 2 + 1;
    typedef IMU::hedata_t sample_t;

    HallEffect(IMU* imu);
    virtual ~HallEffect();


    // rendering functions
    void drawFFT(int width, int height);
    void drawFilt(int width, int height);
//  void drawSig(int width, int height);

    // renders horizontal lines on screen
    void horzLine(float y, int width);

    // changes the hall effect channel displayed in fft form
    void displaySignal(int signum);

    // checks if a channel is being clicked
    bool isClicking(int signum);

    // data manipulation functions
    sample_t getCurrent();
    int getDataLength();
    void clearData();
    std::vector<sample_t> getData();

    // halts data collection thread
    void stop();

private:
    IMU* imu;
    // channel number being displayed as fft
```

```
    int dispsensor = 3;

    std::thread* dataCollectorThread;
    volatile bool running;
    std::mutex siglock, datalock, trainLock;

    // variables related to clicking
    float min[5], max[5], threshpaddingclick, threshpaddingreset;
    volatile bool clicking[5] = {false, false, false, false, false};

    // variables related to data and fft calculations
    sample_t sig[NFFT], filt[NFFT], win[NWIN];
    kiss_fftr_cfg cfg = kiss_fftr_alloc(NFFT,0,0,0);
    kiss_fft_cpx fft[NFFTOUT];
    kiss_fft_cpx filtfft[NFFTOUT];
    std::vector<sample_t> data;
    sample_t current;

    void updateFFT();
    void collectLoop();

};

#endif /* HALLEFFECT_H_ */
```

```
/*
 * HallEffect.cpp
 *
 *  Created on: Jan 8, 2014
 *      Author: snorris
 */

#include "HallEffect.h"

HallEffect::HallEffect(IMU* imu) {
    this->imu = imu;


    threshpaddingclick = .0008;
    threshpaddingreset = 0 ;
    // values taken from measurements
    min[0] = min[1] = min[2] = 0; min[3] = -0.0048842575; min[4] = -0.007448005;
    max[0] = max[1] = max[2] = 1; max[3] = -0.00042773775; max[4] = -0.00256422;
    running = true;
    dataCollectorThread = new std::thread(&HallEffect::collectLoop,this);
}

HallEffect::~HallEffect() {
    delete dataCollectorThread;
}

HallEffect::sample_t HallEffect::getCurrent() {
    return current;
}

int HallEffect::getDataLength(){
    return data.size();
}

std::vector<HallEffect::sample_t> HallEffect::getData() {
    return data;
}

void HallEffect::clearData() {
    datalock.lock();
    data.clear();
    datalock.unlock();
}

void HallEffect::stop() {
    running = false;
    if (dataCollectorThread)
        dataCollectorThread->join();
}

void HallEffect::displaySignal(int signum) {
    siglock.lock();
    dispsensor = signum;
    siglock.unlock();
}

bool HallEffect::isClicking(int signum) {
    bool result;
    siglock.lock();
    result = clicking[signum];

    siglock.unlock();
    return result;
}

void HallEffect::collectLoop() {
    sample_t sample;
    while (running){

        sample = imu->getNextHall();
        if (sample == sample_t::Zero())
            continue; // skip processing if we didn't actually get data

        // subtracts channel 2 (sensor on breadboard) from 3 and 4 (on glove)
        for (int i = 3; i < sample.size(); i++){
            sample[i] -= sample.coeff(2);
        }

        // update data buffers
        siglock.lock();
```

```
        std::rotate(&sig[0],&sig[1],&sig[NFFT]);
        std::rotate(&filt[0],&filt[1],&filt[NFFT]);
        std::rotate(&win[0],&win[1],&win[NWIN]);
        win[NWIN-1] = sig[NFFT-1] = sample;

        // simple short window low pass filter
        filt[NFFT-1] = sample_t::Zero();
        for (int i = 0; i < NWIN; i++){
            filt[NFFT-1] += win[i];
        }
        filt[NFFT-1] /= NWIN;


        current = filt[NFFT-1];


        for (int i = 0; i < 5; i++){
            if (current.coeff(i) > min[i] + threshpaddingreset && current.coeff(i) < max[i] - threshpaddingreset){
                clicking[i] = false;
            }
            else if (current.coeff(i) < min[i]-threshpaddingclick || current.coeff(i) > max[i]+threshpaddingclick){
                clicking[i] = true;
            }
        }
        datalock.lock();
        data.push_back(sample);
        datalock.unlock();

        siglock.unlock();

    }
}

void HallEffect::updateFFT() {
    float tmpsig[NFFT], tmpfilt[NFFT];
    for (int i = 0; i < NFFT; i++){
        tmpsig[i] = sig[i].coeff(dispsensor);
        tmpfilt[i] = filt[i].coeff(dispsensor);
    }

    kiss_fftr(cfg,tmpsig,fft);
    kiss_fftr(cfg,tmpfilt,filtfft);
}

void HallEffect::drawFFT(int width, int height) {
    siglock.lock();
    updateFFT();

    glColor3f(0,0,0);
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < NFFTOUT; i++){
        float mag = 10 * log10f(sqrtf(fft[i].r * fft[i].r + fft[i].i * fft[i].i) / NFFT);
        glVertex2f(((float) i) * width / (NFFTOUT - 1),height / 3 + mag * 5);
    }
    glEnd();

    glColor3f(.5f,.5f,.5f);
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i < NFFTOUT; i++){
        float mag = 10 * log10f(sqrtf(filtfft[i].r * filtfft[i].r + filtfft[i].i * filtfft[i].i) / NFFT);
        glVertex2f(((float) i) * width / (NFFTOUT - 1), height / 3 + mag * 5);
    }
    glEnd();

    siglock.unlock();
}

void HallEffect::drawFilt(int width, int height) {
    siglock.lock();
    sample_t mean = sample_t::Zero();
    for (int i = 0; i < NFFT; i++){
        mean += filt[i];
    }
    mean /= NFFT;


    float scale = 10000;

    glColor3f(0,0,0);
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i< NFFT;i++){
        glVertex2f(((float) i) * width / (NFFT - 1), height / 2 + scale *  (filt[i].coeff(2) - mean.coeff(2)) );
    }
    glEnd();

    if (clicking[3])
        glColor3f(0,0,1);
    else
        glColor3f(0,0,0);
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i< NFFT;i++){
        glVertex2f(((float) i) * width / (NFFT - 1), height / 4 + scale * (filt[i].coeff(3) - mean.coeff(3)) );
    }
    glEnd();

    if (clicking[4])
        glColor3f(0,0,1);
    else
        glColor3f(0,0,0);
    glBegin(GL_LINE_STRIP);
    for (int i = 0; i< NFFT;i++){
        glVertex2f(((float) i) * width / (NFFT - 1), 3 * height / 4 + scale * (filt[i].coeff(4) - mean.coeff(4)) );
    }
    glEnd();


```

```
187     glColor3f(1,0,0);
188     horzLine(height / 4 + scale * (min[3]+threshpaddingreset - mean.coeff(3)) , width);
189     horzLine(height / 4 + scale * (max[3]-threshpaddingreset - mean.coeff(3)) , width);
190
191     horzLine(3 * height / 4 + scale * (min[4]+threshpaddingreset - mean.coeff(4)) , width);
192     horzLine(3 * height / 4 + scale * (max[4]-threshpaddingreset - mean.coeff(4)) , width);
193
194     glColor3f(0,1,0);
195     horzLine(height / 4 + scale * (min[3]-threshpaddingclick - mean.coeff(3)) , width);
196     horzLine(height / 4 + scale * (max[3]+threshpaddingclick - mean.coeff(3)) , width);
197
198     horzLine(3 * height / 4 + scale * (min[4]-threshpaddingclick - mean.coeff(4)) , width);
199     horzLine(3 * height / 4 + scale * (max[4]+threshpaddingclick - mean.coeff(4)) , width);
200
201
202     siglock.unlock();
203 }
204
205 void HallEffect::horzLine(float y, int width){
206     glBegin(GL_LINES);
207     glVertex2f(0,y);
208     glVertex2f(width,y);
209     glEnd();
210 }
211
212 //void HallEffect::drawSig(int width, int height) {
213 //  siglock.lock();
214 //  glColor3f(1,0,0);
215 //  double mean = 0;
216 //  for (int i = NFFT * 3 / 4; i< NFFT;i++){
217 //      mean+= sigx[i];
218 //  }
219 //  mean = mean * 4 / NFFT;
220 //
221 //  glBegin(GL_LINE_STRIP);
222 //  for (int i = NFFT * 3 / 4; i< NFFT;i++){
223 //      glVertex2f(((float) i - NFFT * 3 / 4) * width / (NFFT / 4 - 1), height / 2 + 1000 * (sigx[i] - mean));
224 //  }
225 //  glEnd();
226 //
227 //  glColor3f(0,1,0);
228 //  mean = 0;
229 //  for (int i = NFFT * 3 / 4; i< NFFT;i++){
230 //      mean+= sigy[i];
231 //  }
232 //  mean = mean * 4 / NFFT;
233 //  glBegin(GL_LINE_STRIP);
234 //  for (int i = NFFT * 3 / 4; i< NFFT;i++){
235 //      glVertex2f(((float) i - NFFT * 3 / 4) * width / (NFFT / 4 - 1), height / 2 + 1000 * (sigy[i] - mean));
236 //  }
237 //  glEnd();
238 //
239 //  glColor3f(0,0,1);
240 //  mean = 0;
241 //  for (int i = NFFT * 3 / 4; i< NFFT;i++){
242 //      mean+= sigz[i];
243 //  }
244 //  mean = mean * 4 / NFFT;
245 //  glBegin(GL_LINE_STRIP);
246 //  for (int i = NFFT * 3 / 4; i< NFFT;i++){
247 //      glVertex2f(((float) i - NFFT * 3 / 4) * width / (NFFT / 4 - 1), height / 2 + 1000 * (sigz[i] - mean));
248 //  }
249 //  glEnd();
250 //  siglock.unlock();
251 //}
```

A.2 IMU processing object

This object abstracts the IMU processing, starting a connection with the IMU, but actually communicating with the microprocessor via a modified vectornav 100 library. The modified library includes code to recognize the hall effect data and pass it along through the same channels to the IMU, which in turns provides the data to the HallEffect object. The IMU object receives data asynchronously and processes each raw data sample on receipt.

```
1 /*
2  * IMU.h
3  *
4  *  Created on: Sep 12, 2013
5  *      Author: stephennorris
6  */
7
8 #ifndef IMU_H_
```

```
#define IMU_H_
#include <string>
#include <mutex>
#include <iostream>
#include "vectornav/vectornav.h"
#include <eigen3/Eigen/Dense>
#include <math.h>
#include <sys/time.h>
#include <unistd.h>

class IMU {
public:
    typedef Eigen::Matrix<float,1,5> hedata_t;
private:

    const char* COM_PORT;
    int BAUD_RATE;
    unsigned int dataOutputType;

    // used to calculate Y inputs
    const double pitchdist = 1 / tan(M_PI / 180);

    // Filter design to be used
    int filterMode = 0;

    int sampleCount = 0;

    VN_ERROR_CODE error;
    std::mutex hallLock, dataLock, pointLock;
    Vn100CompositeData rawData;
    hedata_t rawhedata;

    volatile bool newMag = false, newHallReady = false, switchon = false;
    Eigen::Vector3d lastPointer;
    Eigen::Vector2d pos, diffPos, lastPos;
    Eigen::Matrix3d zeroRot, openGLCvt;

    // Variables related to friction simulation
    const static int FRICT_EST_WIN = 25;
    typedef Eigen::Matrix<double,1,FRICT_EST_WIN> frictv;
    frictv frictSig;
    double frictDataX[FRICT_EST_WIN], frictDataY[FRICT_EST_WIN];
    bool stuck = false;
    double UNSTICK = pow(10, -0.5) / 36.0555, STICK = pow(10, -0.9) / 36.0555;
    Eigen::Vector2d updateFrict();

    // Used for a detector to wake the cursor up
    const static int SHAKE_WIN = 15;
    double diffXPoint[SHAKE_WIN];
    int leftImpulse[2] = {-1000, -1000}, rightImpulse[2] = {-1000, -1000};

    // Record of pointing data
    std::vector<Eigen::Vector2d> data;

    // Sensitivity curve variables
    Eigen::Matrix2d C;
    Eigen::Vector2d mu;
    bool isNoise(const Eigen::Vector2d& X);
    Eigen::Vector2d dXEstimate(Eigen::Vector2d dX);

    // Used for communicating with microprocesser/IMU
    static IMU* vn100Listener;
    static void dataReciever(Vn100* sender, Vn100CompositeData* newData);
    static void hallReciever(const float hedata[5], int on);

    // Update functions for data
    void newData(Vn100CompositeData* newData);
    void updatePointing(Vn100CompositeData* newData);
    void updateWake(Eigen::Vector2d & dRawPoint);
    void newHall(const float hedata[5], int on);


public:
    IMU(unsigned int dataOutputType = VNASYNC_VNQTN, std::string port = "/dev/ttyACM0", int rate = 921600);
    virtual ~IMU();

    Vn100 sensor;
    // IMU interfacing functions
    void printVPEinfo();
    void setVPE(unsigned char enable, unsigned char heading, unsigned char  filter, unsigned char  tuning);
    double getFrequency();
    void setFrequency(unsigned int freq);
    // Returns a readable error from the last IMU response
    std::string getError();


    // Data recording functions
    void clearData();
    int getDataLength();
    std::vector<Eigen::Vector2d> getData();


    // returns the current raw data object
    void getData(Vn100CompositeData& data);
    // return various forms of the current data (when available)
    hedata_t getHallEffect();
    double getPointingRoll();
    double getPointingPitch();
    Eigen::Vector2d getPosition();
    Eigen::Vector2d getDiffPosition();
    Eigen::Matrix3d getRotMat();
    Eigen::Matrix3d getRawRotMat();
    Eigen::Matrix3d getOpenGLRotMat();
    Eigen::Matrix3d getOpenGLRawRotMat();
    // return data, but block until a new point is available
    Eigen::Vector3d getNextMag();
```

```
    hedata_t getNextHall();


    // returns state of switch on breadboard
    bool getSwitchOn();
    // returns state of a detector used to wake up the cursor
    bool getWake();


    // functions that affect IMU processing
    void tare();
    void setFilterMode(int mode);
    void setZeroRot(Eigen::Matrix3d rot);
};



#endif /* IMU_H_ */
```

```
/*
 * IMU.cpp
 *
 *  Created on: Sep 12, 2013
 *      Author: stephennorris
 */

#include "IMU.h"

IMU* IMU::vn100Listener = NULL;

IMU::IMU( unsigned int dataOutputType, std::string port, int rate) {
    this->dataOutputType = dataOutputType;
    C << .4610, 0, 0, .6216;
    C /= 10000.0;
    C = C.inverse().eval();
    mu << 0, 0;

    // use for unstick 2*sigma_x
    UNSTICK = 2 * sqrt(1 / C.coeff(0));

    lastPointer = Eigen::Vector3d::Zero();
    pos = diffPos = lastPos = Eigen::Vector2d::Zero();
    zeroRot = Eigen::Matrix3d::Zero();
    openGLCvt << 0, 1, 0,
                 0, 0, 1,
                 1, 0, 0;

    for (int i = 0; i < FRICT_EST_WIN; i++){
        frictDataX[i] = frictDataY[i] = i;
    }
    frictSig = Eigen::VectorXd::LinSpaced(FRICT_EST_WIN,-FRICT_EST_WIN / 2,FRICT_EST_WIN / 2);
    frictSig = frictSig / (frictSig * frictSig.transpose());


    COM_PORT = port.c_str();
    BAUD_RATE = rate;

    vn100Listener = this;

    error = vn100_connect(&sensor, COM_PORT, BAUD_RATE);
    if (error){
        std::cout << "Error while connecting: " << getError() << std::endl;
    }
    vn100_reset(&sensor);
    usleep(100000);
    error = vn100_setAsynchronousDataOutputType(&sensor,dataOutputType, true);
//  error = vn100_setAsynchronousDataOutputType(&sensor,VNASYNC_VNMAG, true);
//  error = vn100_setAsynchronousDataOutputType(&sensor,VNASYNC_VNQTN, true);
//  error = vn100_setAsynchronousDataOutputType(&sensor,VNASYNC_VNYPR, true);
    if (error){
        std::cout << "Error setting data output type: " << getError() << std::endl;
    }

    error = vn100_registerAsyncDataReceivedListener(&sensor,IMU::dataReciever,IMU::hallReciever);
    if (error){
        std::cout << "Error registering async output: " << getError() << std::endl;
    }

    std::cout << "Starting IMU..." << std::endl;


}

IMU::~IMU() {
    vn100_unregisterAsyncDataReceivedListener(&sensor, IMU::dataReciever, IMU::hallReciever);
    error = vn100_disconnect(&sensor);
    if (error){
        std::cout << "Error while disconnecting: " << getError() << std::endl;
    }
}

std::string IMU::getError() {
    switch (error){
    case VNERR_NO_ERROR:
        return 0;
    case VNERR_UNKNOWN_ERROR:
        return "Unknown error.";
    case VNERR_FILE_NOT_FOUND:
        return "File not found. (incorrect /dev/*)";
    case VNERR_INVALID_VALUE:
        return "Invalid Value.";
    case VNERR_TIMEOUT:
        return "Timeout.";
    case VNERR_NOT_CONNECTED:
        return "Not connected.";
```

```
87      case VNERR_NOT_IMPLEMENTED:
88          return "Not implemented.";
89      default:
90          return "Unspecified error.";
91      }
92
93  }
94
95  bool IMU::isNoise(const Eigen::Vector2d& X){
96      return - 2 * log(.00001) > (X - mu).transpose() * C * (X - mu);
97  }
98
99  Eigen::Vector2d IMU::dXEstimate(Eigen::Vector2d dX){
100     double rsq = ((dX - mu).transpose() * C * (dX - mu));
101     return (1 - exp(- rsq / 2.0)) * dX;
102 }
103
104 Eigen::Vector2d IMU::getPosition() {
105
106     Eigen::Vector2d result;
107     pointLock.lock();
108     result = pos;
109     pointLock.unlock();
110     return result;
111 }
112
113 Eigen::Vector2d IMU::getDiffPosition() {
114     Eigen::Vector2d result;
115     pointLock.lock();
116     result = diffPos;
117     pointLock.unlock();
118     return result;
119 }
120
121 void IMU::tare() {
122
123     pointLock.lock();
124     Eigen::Matrix3d rot = getRawRotMat();
125     setZeroRot(rot);
126
127     lastPointer = getRotMat().col(0);
128
129     for (int i = 0; i < SHAKE_WIN; i++)
130         diffXPoint[i] = 0;
131
132     pointLock.unlock();
133
134 }
135
136 void IMU::updatePointing(Vn100CompositeData* newData){
137     pointLock.lock();
138
139     Eigen::Vector3d curPointer = getRotMat().col(0);
140     double newyaw = 180 * atan2(curPointer.coeff(1) , curPointer.coeff(0)) / M_PI,
141            oldyaw = 180 * atan2(lastPointer.coeff(1) , lastPointer.coeff(0)) / M_PI,
142            pitch = pitchdist * curPointer.coeff(2) / sqrt( pow(curPointer.coeff(1),2)+pow(curPointer.coeff(0),2 ) )
       ;
143     pos(0) = lastPos.coeff(0) + fmod(newyaw - oldyaw +540.0, 360.0) - 180.0;
144     pos(1) = -pitch;
145
146
147     lastPointer = curPointer;
148
149     Eigen::Vector2d rawdPos = pos - lastPos;
150
151     data.push_back(pos);
152
153     updateWake(rawdPos);
154
155
156
157     // 0 = full design, 1 = raw, 2 = a deadzone only design, 3 = sensitivity curve only
158     if (filterMode == 0){
159         diffPos = updateFrict();
160     } else if (filterMode == 2){
161         if (isNoise(rawdPos)){
162             diffPos = Eigen::Vector2d::Zero();
163             std::cout << "Noise..." << std::endl;
164         }else{
165             diffPos = rawdPos;
166             std::cout << "Wasn't noise..." << std::endl;
167         }
168     } else if (filterMode == 3){
169         diffPos = dXEstimate(rawdPos);
170     }else{
171         diffPos = rawdPos;
172     }
173
174     lastPos = pos;
175
176     pointLock.unlock();
177 }
178
179 Eigen::Vector2d IMU::updateFrict() {
180     std::rotate(&frictDataX[0],&frictDataX[1],&frictDataX[FRICT_EST_WIN]);
181     std::rotate(&frictDataY[0],&frictDataY[1],&frictDataY[FRICT_EST_WIN]);
182
183     frictDataX[FRICT_EST_WIN - 1] = pos.coeff(0);
184     frictDataY[FRICT_EST_WIN - 1] = pos.coeff(1);
185
186     frictv frictVx = frictv(frictDataX);
187     frictv frictVy = frictv(frictDataY);
188     double Vx = frictVx * frictSig.transpose(),
189            Vy = frictVy * frictSig.transpose();
190     double V = sqrt(Vx * Vx + Vy * Vy);
191
```

```
192     if (stuck && V > UNSTICK){
193         stuck =false;
194         std:: cout << "Unstuck!" << std::endl;
195     }
196     else if (!stuck && V < STICK){
197         stuck = true;
198         std::cout << "Sticking!" << std::endl;
199     }
200 
201     Eigen::Vector2d dRawPos = pos - lastPos;
202     return stuck ? Eigen::Vector2d::Zero() : dXEstimate(dRawPos);
203 }
204 
205 void IMU::updateWake(Eigen::Vector2d & dRawPoint){
206     std::rotate(&diffXPoint[0],&diffXPoint[1],&diffXPoint[SHAKE_WIN]);
207     diffXPoint[SHAKE_WIN-1] = dRawPoint[0];
208 
209     double maxImpulse = 0;
210     int impIndex = 0;
211     for (int n = 0; n < SHAKE_WIN - 2; n++){
212         for (int dist = 1; dist < SHAKE_WIN-1 - n; dist++){
213             double diffx = diffXPoint[n + dist] - diffXPoint[n];
214             if (fabs(diffx) > fabs(maxImpulse)){
215                 maxImpulse = diffx;
216                 impIndex = n;
217             }
218         }
219     }
220 
221     rightImpulse[0]--;
222     rightImpulse[1]--;
223     leftImpulse[0]--;
224     leftImpulse[1]--;
225 
226     double impthresh = 1.25;
227     if (maxImpulse > impthresh){
228         if (impIndex != rightImpulse[0] && impIndex != rightImpulse[1]){
229             rightImpulse[0] = rightImpulse[1];
230             rightImpulse[1] = impIndex;
231         }
232     }else if (maxImpulse < -impthresh){
233         if (impIndex != leftImpulse[0] && impIndex != leftImpulse[1]){
234             leftImpulse[0] = leftImpulse[1];
235             leftImpulse[1] = impIndex;
236         }
237     }
238 
239 }
240 
241 void IMU::newData(Vn100CompositeData* newData) {
242     dataLock.lock();
243     newMag = true;
244     rawData = *newData;
245     dataLock.unlock();
246 
247     if (sampleCount == 0){
248         setZeroRot(getRawRotMat());
249     }
250     sampleCount = sampleCount < 300? sampleCount + 1 : sampleCount;
251 
252 
253     updatePointing(newData);
254 
255 }
256 
257 void IMU::getData(Vn100CompositeData& data) {
258     dataLock.lock();
259     data = rawData;
260     dataLock.unlock();
261 }
262 
263 double IMU::getPointingRoll(){
264     Eigen::Vector3d pointDir = getRotMat().col(0), // Forwards (Pointing) Vector
265                     upDir = getRotMat().col(2); // Up Vector
266     pointDir[2] = 0; // Flatten Forwards Vector into horizontal plane
267     pointDir.normalize();
268     upDir -= upDir.dot(pointDir) * pointDir; // Gramm-Schmidt process, remove component of upDir parallel to
      pointing
269 
270     Eigen::Vector2d col1 = {pointDir[0], pointDir[1]}, col2 = {upDir[0], upDir[1]};
271     col1.normalize();
272     col2.normalize();
273     Eigen::Matrix2d signFinder;
274     signFinder << col1, col2;
275 
276 
277     double x = upDir[0], y = upDir[1], z = upDir[2];
278     return 180.0 * atan2(signFinder.determinant() * sqrt(x * x + y * y),z) / M_PI;
279 }
280 
281 double IMU::getPointingPitch(){
282     double result;
283     pointLock.lock();
284     result = 180 * atan(  lastPointer.coeff(2) / sqrt( pow(lastPointer.coeff(1),2) + pow(lastPointer.coeff(0),2) )
      ) / M_PI;
285     pointLock.unlock();
286     return result;
287 }
288 
289 std::vector<Eigen::Vector2d> IMU::getData() {
290     return data;
291 }
292 
293 void IMU::clearData() {
294     pointLock.lock();
295     data.clear();
```

```
296     pointLock.unlock();
297 }
298
299 int IMU::getDataLength(){
300     return data.size();
301 }
302
303 void IMU::setZeroRot(Eigen::Matrix3d rot) {
304     Eigen::Vector3d xVec(rot(0,0),rot(1,0), 0), zVec(0,0,1);
305     xVec.normalize();
306     Eigen::Vector3d yVec(rot(0,1),rot(1,1), 0);
307     yVec -= yVec.dot(xVec) * xVec;
308     yVec(2) = 0;
309     yVec.normalize();
310     double cosx = xVec[0];
311     std::cout << "Angle: " << acos(cosx) << std::endl;
312     zeroRot.col(0) = xVec;
313     zeroRot.col(1) = yVec;
314     zeroRot.col(2) = zVec;
315     zeroRot = zeroRot.inverse().eval();
316 }
317
318 Eigen::Matrix3d IMU::getRotMat() {
319     return zeroRot * getRawRotMat();
320 }
321
322 Eigen::Matrix3d IMU::getRawRotMat(){
323     Eigen::Matrix3d result;
324     dataLock.lock();
325     VnQuaternion &rot = rawData.quaternion;
326     Eigen::Quaterniond rotation(rot.w,rot.x,rot.y,rot.z);
327     dataLock.unlock();
328     result = rotation.matrix();
329     return result;
330 }
331 Eigen::Matrix3d IMU::getOpenGLRotMat() {
332     Eigen::Matrix3d result = openGLCvt * getRotMat() * openGLCvt.transpose();
333     result.col(0) = result.col(0) * -1;
334
335     return result;
336 }
337
338 Eigen::Matrix3d IMU::getOpenGLRawRotMat() {
339     Eigen::Matrix3d result = openGLCvt * getRawRotMat() * openGLCvt.transpose();
340     return result.transpose();
341 }
342
343 void IMU::printVPEinfo() {
344     unsigned char enable, heading, filter, tuning;
345     vn100_getVpeControl(&sensor,&enable,&heading,&filter,&tuning);
346
347     std::cout << "Enabled: " << (int) enable << ", heading: " << (int) heading << ", filter: " << (int) filter << ",
        tuning: " << (int) tuning << std::endl;
348 }
349
350 void IMU::setVPE(unsigned char enable, unsigned char heading, unsigned char  filter, unsigned char  tuning){
351     vn100_setVpeControl(&sensor,enable,heading,filter,tuning,true);
352 }
353
354 double IMU::getFrequency() {
355     unsigned int freq;
356     vn100_getAsynchronousDataOutputFrequency(&sensor,&freq);
357     return freq;
358 }
359
360 void IMU::setFrequency(unsigned int freq) {
361     vn100_setAsynchronousDataOutputFrequency(&sensor, freq, true);
362 }
363
364 void IMU::setFilterMode(int mode) {
365     pointLock.lock();
366     filterMode = mode;
367     pointLock.unlock();
368 }
369
370 bool IMU::getSwitchOn() {
371     bool result = false;
372     hallLock.lock();
373     result = switchon;
374     hallLock.unlock();
375     return result;
376 }
377
378 bool IMU::getWake() {
379     int timeout = (-5 * SHAKE_WIN);
380     return rightImpulse[0] > timeout && rightImpulse[1] > timeout && leftImpulse[0] > timeout && leftImpulse[1] >
        timeout;
381 }
382
383 IMU::hedata_t IMU::getHallEffect() {
384     hedata_t result;
385     hallLock.lock();
386     result = rawhedata;
387     hallLock.unlock();
388     return result;
389 }
390
391 IMU::hedata_t IMU::getNextHall(){
392     unsigned int waitCount = 0;
393     hallLock.lock();
394     while(!newHallReady){
395         hallLock.unlock();
396         usleep(1000);
397         waitCount++;
398         if (waitCount > 1000)
399             return hedata_t::Zero();
```

```
400         hallLock.lock();
401     }
402     hedata_t result = rawhedata;
403     newHallReady = false;
404     hallLock.unlock();
405     return result;
406 }
407
408 void IMU::newHall(const float hedata[5], int on){
409     hallLock.lock();
410     newHallReady = true;
411     switchon = (bool) on;
412     for (int i =0; i < 5; i++){
413         rawhedata[i] = hedata[i];
414     }
415     hallLock.unlock();
416 }
417
418 void IMU::hallReciever(const float hedata[5], int on){
419     vn100Listener->newHall(hedata, on);
420 }
421
422 void IMU::dataReciever(Vn100* sender, Vn100CompositeData* newData){
423     vn100Listener->newData(newData);
424 }
425
426 Eigen::Vector3d IMU::getNextMag() {
427     unsigned int waitCount = 0;
428     dataLock.lock();
429     while (!newMag){
430         dataLock.unlock();
431         usleep(1000);
432         waitCount++;
433         if (waitCount > 1000)
434             return Eigen::Vector3d::Zero();
435         dataLock.lock();
436     }
437     Eigen::Vector3d result = {rawData.magnetic.c0, rawData.magnetic.c1, rawData.magnetic.c2};
438     newMag = false;
439     dataLock.unlock();
440     return result;
441 }
```

APPENDIX B

Source code for LPC1768 microprocessor

This is the code used on the microprocessor for sampling the hall effect sensors and also passing communication from the computer to the IMU and vice-versa.

```
#include "mbed.h"
#include <algorithm>
#define M_PI 3.14159265358979323846
#include "math.h"

DigitalOut myled[4] = {DigitalOut(LED1),DigitalOut(LED2),DigitalOut(LED3),DigitalOut(LED4)};
DigitalIn on = DigitalIn(p21);
RawSerial pc(USBTX, USBRX); // tx, rx
RawSerial vn(p13,p14);
AnalogIn analog[5] = {AnalogIn(p15), AnalogIn(p16), AnalogIn(p17), AnalogIn(p19), AnalogIn(p20)};

float analogsamples[5];
bool newAnalog = false;

int prev, lastDiff;
Timer timer;
void getSample()
{
    if (newAnalog)
        myled[3] = 1;
    else
        myled[3] = 0;
    for (int i = 0; i < 5; i++)
        analogsamples[i] = analog[i].read();
    int now = timer.read_us();

    lastDiff = now-prev;
    newAnalog = true;

    prev = now;
}


int main(){
    pc.baud(921600);
    vn.baud(115200);

    Ticker tick;
    tick.attach(&getSample, 0.01f);
    timer.start();
    prev = timer.read_us();

    int charsleft = -1;

    while(1) {
        if (pc.readable()){
            vn.putc(pc.getc());
        }
        if (vn.readable()){
            int nextc = vn.getc();

            if (charsleft >= 0)
                charsleft--;

            if (nextc == '*'){
                charsleft = 4;
            }

            pc.putc(nextc);
        }

        if (charsleft == 0 && newAnalog){
            pc.printf("$MAGIC,%i,%i,%f,%f,%f,%f,%f*FF\r\n",lastDiff,on.read(),analogsamples[0],analogsamples[1],
      analogsamples[2],analogsamples[3],analogsamples[4]);
            newAnalog = false;
        }
    }
}
```

BIBLIOGRAPHY

"Human benchmark." Mar. 2016. [Online]. Available: http://www.humanbenchmark.com/

Bajramovic, M., "Computer mouse on a glove," July 2003, uS Patent App. 10/382,849. [Online]. Available: http://www.google.com/patents/US20030137489

Bogdanoff, D. "The touch screen revolution." Feb. 2015. [Online]. Available: http://www.techbriefs.com/component/content/article/ntb/features/feature-articles/21545

Delsys Incorporated. "Trigno lab — wireless emg system." 2015. [Online]. Available: http://www.delsys.com/products/wireless-emg/trigno-lab/

Forbes, T., "Mouse hci through combined emg and imu," Master's thesis, University of Rhode Island: Open Access Master's Theses Paper 43, 2013. [Online]. Available: http://digitalcommons.uri.edu/theses/43

Haque, F., Nancel, M., and Vogel, D., "Myopoint: Pointing and clicking using forearm mounted electromyography and inertial motion sensors," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 3653–3656. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702133

Kay, S. M., *Fundamentals of Statistical Signal Processing: Estimation Theory*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

Lakie, M., Vernooij, C. A., Osborne, T. M., and Reynolds, R. F., "The resonant component of human physiological hand tremor is altered by slow voluntary movements," *The Journal of Physiology*, vol. 590, no. 10, pp. 2471–2483, 2012. [Online]. Available: http://dx.doi.org/10.1113/jphysiol.2011.226449

NXP. "Arm mbed lpc1768 board." 2015. [Online]. Available: http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/lpc-cortex-m-mcus/lpc-cortex-m3/lpc1700-series/arm-mbed-lpc1768-board:OM11043

Riviere, C., Rader, R. S., and Thakor, N. V., "Adaptive canceling of physiological tremor for improved precision in microsurgery," *IEEE Transactions on Biomedical Engineering*, vol. 45, no. 7, pp. 839 – 846, July 1998.

Rubchinsky, L. L., Kuznetsov, A. S., MD, V. L. W., and Sigvardt, K. A., "Tremor," *Scholarpedia*, vol. 2, no. 10, p. 1379, 2007, revision 135551.

Sean Chen, E. L. Cornell University. "Mister gloves - a wireless usb gesture input system." 2010. [Online]. Available: https://courses.cit.cornell.edu/ee476/FinalProjects/s2010/ssc88_egl27/

Thalmic Labs. "Myo gesture control armband - wearable technology by thalmic labs." 2015. [Online]. Available: https://www.myo.com/

VectorNav. "Vn-100 rugged." 2015. [Online]. Available: http://www.vectornav.com/products/vn100-rugged

Xiong, A., Chen, Y., Zhao, X., Han, J., and Liu, G., "A novel hci based on emg and imu," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, Dec 2011, pp. 2653–2657.